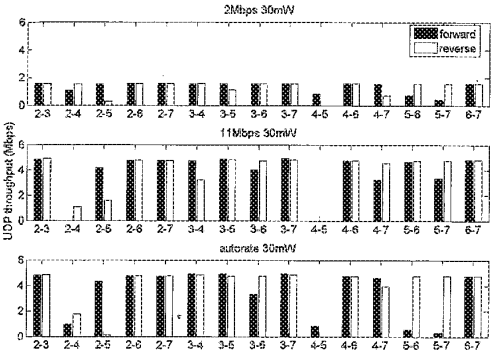


Fig. 15. Effect of autorate for *ukhome1* at *txpower=30mW* (*ukhome1 – layout1 – udp – autorate*). Average throughput in Mbps top to bottom: 1.31, 3.84, 3.73.

In such an environment, network topology can have a significant impact on network performance. In this section, we use the results from Sections III and IV to evaluate the impact of three possible topologies: AP-based communication, direct communication, and multi-hop (or mesh) communication.

A. Topologies

Today most home wireless networks utilize an AP-based topology. Each packet originating from a node in the wireless network is first transmitted to an access point. The AP then forwards the packet to the destination, which may be located in the wireless network or may be accessible through some other (typically wired) interface. This topology is reasonable (though not necessarily optimal) when most traffic flows between wireless stations and the Internet through an AP.

AP-based topologies are less optimal for traffic that flows between stations within the wireless network. And, as the number of wireless-enabled devices in the home increases the amount of intra-wireless-network traffic will also increase. Examples of such traffic include computer-to-printer traffic, as well as communication between consumer electronic devices. Such traffic pays a two-hop penalty, since all packets transmitted by the source must be retransmitted by the AP, and the AP contends for the same channel as the originating node when forwarding packets. This approach is particularly sub-optimal if the communicating devices are relatively close, while the AP is relatively far away. Moreover, given our evaluation of wireless link characteristics, high-bandwidth communication in home networks between the source and destination nodes and the AP are not assured. In addition, choosing a location for an AP that maximizes network throughput is non-trivial.

Direct communication between wireless stations is an alternative to AP-based communication that eliminates the two-hop penalty. These topologies are most relevant for traffic between two devices within the home. However, our results from the previous sections demonstrate that home networks may not be able to guarantee the interaction between every two devices in the home.

Mesh networking provides a third alternative to direct and AP-based communication. Mesh networks allow any node to forward packets on behalf of other nodes. Thus, a wide variety of different topologies are possible. Typically a routing algorithm is used to evaluate various paths between each pair of communicating nodes and selects the one that optimizes a particular metric. In particular, mesh networking can allow direct communication between nodes, two-hop communication through a third node, or more hops when necessary. While each hop introduces an additional bandwidth penalty (since each hop contends for the same channel, as in the AP-based case), multiple high-quality links may be better than a single low-quality link. Previous work has shown that such multi-hopping strategies could offer higher throughput multi-hop connections, when the quality of the direct link is poor [13]. Multi-hopping mechanisms and protocols are currently being explored within the IEEE 802.11s task group [7].

In Figure 16 we present an example node layout to demonstrate schematically the topologies considered in this section. Each link is labelled with the link loss rate, as might be measured from our reachability test. Figure 16(a) demonstrates the "direct" topology that could be employed for the communication between nodes; every node communicates with every other node using the direct path. Note that direct communication between node-6 and node-7 has a 80% loss rate in this example. Figure 16(b) presents an AP-based topology, with node-5 acting as the AP. In this case, communication between node-6 and node-7, through node-5 achieves a 90% loss rate. Lastly, Figure 16(c) presents an alternative topology, based on multi-hopping. If routing is determined using Dijkstra's algorithms and link weights reflect loss rate, then the optimal routing for node-6 to reach every other node inside the network is shown with dark lines. In this case, node-6 can reach node-7 through node-2 with 100% success.

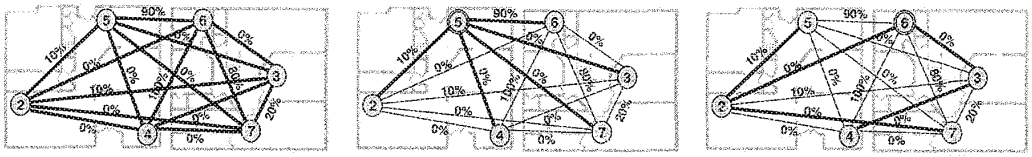
The question we address in the remainder of this section is the impact of topology on home wireless network performance. If, for example, a mesh topology could significantly increase overall throughput in home networks, a change in the design and functionality of radios for consumer electronics would be warranted.

B. Methodology

To assess the potential gain from alternative topologies in the home, we need a means to map the throughput values measured through our experiments to the throughput that would be achieved by different node pairs under alternative topologies. The fundamental difference between our measurements and the throughput that would be achieved under the AP or mesh topology is that specific nodes now use intermediate nodes for their communication. The throughput achieved by node-A to node-C through node-B can be estimated using the airtime consumed by each individual transmission. For this purpose, we make the pessimistic assumption that all nodes are within the same collision domain.

In our current measurements, the airtime consumed by node-A's transmission to node-C is the inverse of the throughput of the link from node-A to node-C, or $\frac{1}{thr_{A,C}}$, which is the amount of time the wireless channel is busy. If node-A uses node-B as a relay, then an individual transmission now leads to

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1149
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



(a) direct communication (b) AP topology - AP at *node-5* (c) mesh topology

Fig. 16. Example topologies for the interactions sourcing at *node-6*. Link weights represent loss rates along the direct path.

two transmissions: one from node-A to node-B, and one from node-B to node-C, with respective airtimes of $\frac{1}{thr_{A,B}}$, $\frac{1}{thr_{B,C}}$. Consequently, the throughput that would be observed by node-A toward node-C through AP node-B can be expressed as:

$$thr_{S,D} = \frac{1}{\frac{1}{thr_{S,A}} + \frac{1}{thr_{A,D}}} \quad (1)$$

The above formula describes the airtime for a transmission between node-A and node-C through node-B as the sum of the air times for the transmissions between node-A and node-B, and node-B and node-C.

We validate this formula experimentally and use it to assess the impact of alternative topologies in the studied homes. For the validation step we measure the multihop effect as observed from node-2 in *ukhome1* when *txrate* is 2Mbps or 11Mbps. Our experiments cover the full set of combinations for destination and relay nodes, and results are presented in Table II for the case in which *txrate=11Mbps*. Each experiment measures the throughput of the two single-hop paths comprising the multi-hop path. Then the source and relay nodes are configured appropriately to facilitate multihop communication, and the end-to-end throughput is measured. Note that the experimental measurement of the two single-hop throughputs preceded the measurement of the multihop throughput, and therefore slight variations should be expected.

The absolute and relative errors for *txrate=11Mbps* are shown in Table II. The absolute error (the magnitude difference between the measured and estimated values in Mbps) incurred using Eq. 1 is small. The distribution of the relative errors (the percentage difference between the measured and estimated values) incurred across both rates and for all the possible multihop connections initiating from *node-2* is further shown in Figure 17. We notice that 80% of the relative error values obtained when *txrate=2Mbps* are below 5%. The respective number for *txrate=11Mbps* is slightly greater (10%), which is a direct consequence of the fact that a slight change in the conditions across the measurement of $thr_{A,B}$, $thr_{B,C}$, $thr_{A,C}$ has a more significant impact. Given the results in Table II and Figure 17, we conclude that the "airtime" metric is appropriate for the investigation of the performance implications of alternative topologies in the home.

TABLE II

MULTIHOP THROUGHPUT FROM NODE-2 TO NODE-*dst* THROUGH NODE-*rtr* WHEN *txrate=11Mbps*, *txpower=30mW* (*ukhome1 -- layout1 -- udp -- multihop*).

dst	rtr	1st hop (*Mbps*)	2nd hop (*Mbps*)	multihop (*Mbps*)	estimate (*Mbps*)	abserr (*Mbps*)	relerr (%)
3	4	5.04	6.29	3.14	2.79	0.35	10
3	5	5.18	2.93	1.83	1.87	0.04	2
3	6	5.82	6.30	3.05	3.02	0.02	0
3	7	6.11	6.27	3.10	3.09	0.00	0
4	3	6.25	6.08	3.34	3.08	0.25	7
4	5	5.07	5.49	3.14	2.63	0.50	16
4	6	6.02	6.15	3.19	3.04	0.14	4
4	7	6.11	6.12	2.57	3.05	0.48	18
5	3	6.26	5.20	2.91	2.84	0.06	2
5	4	5.57	4.14	2.28	2.37	0.09	4
5	6	5.99	5.86	3.06	2.96	0.09	3
5	7	6.11	5.15	2.37	2.79	0.42	17
6	3	6.26	6.12	3.24	3.09	0.14	4
6	4	5.57	6.15	3.17	2.92	0.24	7
6	5	5.02	3.63	2.09	2.10	0.01	0
6	7	6.11	5.65	3.25	2.93	0.31	9
7	3	6.26	6.13	3.23	3.09	0.13	4
7	4	5.41	6.15	3.13	2.87	0.25	8
7	5	5.01	4.60	2.60	2.39	0.20	7
7	6	5.95	6.05	2.97	2.99	0.02	1

C. Findings

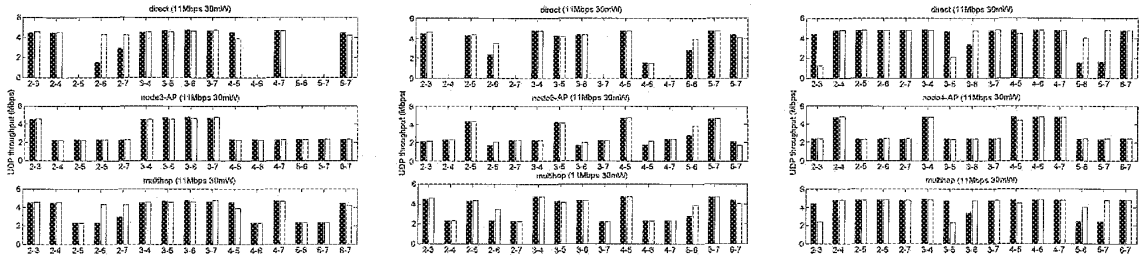
Previously we conjectured that random selection of the location of the Access Point (AP) in the home network is not guaranteed to lead to a high performance wireless network. Using the collected throughput measurements we quantify the throughput achievable by different pairs of communicating nodes, when a particular testbed node is selected to act as an AP. For instance, assuming that the AP is located at the position of node-2 we quantify the throughput achieved from node-X to node-Y given that it has to multihop through node-2. Our results are presented in Figure 18.

Figure 18 clearly demonstrates the impact of the location of the AP on the network performance. Only one of the six node locations in the testbed (node-3) leads to a network in which no pair of nodes experiences zero throughput. For any other choice of AP location (of the six possible locations), 35%-80% of the links are unusable (i.e. zero throughput).

Next we compare the network performance in each home for the direct topology, the best AP topology, and the optimal multi-hop topology. Note that the best topology for each home is determined using the same method shown in Figure 18. Multihop routing is derived using Dijkstra's algorithm, where link weights are proportional to measured loss rate. While this routing scheme determines optimal routes, it is important to note that it does not necessarily provide an optimal routing algorithm. The reader interested in mesh routing can refer to [6] and references therein.

Results for each of *ushome1*, *ushome2*, and *ukhome1* are presented in Figure 19 and brief overall summaries are given

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1150
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



(a) (b) (c)

Fig. 19. The impact of different topologies on UDP throughput (a) *ushome1*, (b) *ushome2* and (c) *ukhome1* (*txrate=11Mbps*, *txpower=30mW*, $* - layout1 - udp$). Throughput statistics can be found in Table III.

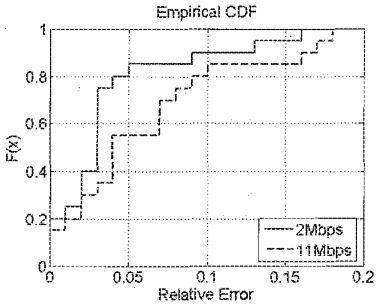


Fig. 17. Relative error in the estimation of multihop throughput ($ukhome1 - layout1 - udp - multihop$).

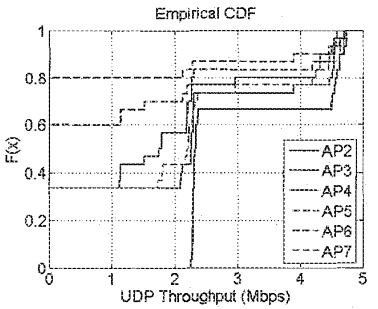


Fig. 18. The impact of AP location (*txrate=11Mbps*, *txpower=30mW*, $ushome1 - layout1 - udp - multihop$).

TABLE III

PERFORMANCE STATISTICS FROM ALL THREE HOMES UNDER ALTERNATIVE TOPOLOGIES ($* - layout1 - udp - multihop$).

Home	Metric	Direct	best AP-topology	Multihop
ushome1	Minimum Thr.	0	0	2.2
	Average Thr.	3.15	3.07	3.79
	Aggregate Thr.	94.58	92.26	113.88
ushome2	Minimum Thr.	1.23	1.14	2.3
	Average Thr.	2.89	2.81	3.55
	Aggregate Thr.	86.88	84.43	106.61
ukhome1	Minimum Thr.	0	0	2.2
	Average Thr.	4.24	3.16	4.35
	Aggregate Thr.	127.36	94.88	130.41

in Table III. The AP-based topology performs the worst, with lower average and aggregate throughput in nearly every case, and lower minimum throughput than both direct and multihop topologies in *ushome2*. As expected, the direct communication topology does eliminate the two-hop penalty (except in the case that either the source or destination is the AP). Thus, many additional node pairs are able to achieve the highest possible throughput. However, the direct topology also disables communication between many node pairs. The mesh topology neither improves nor harms the throughput of the links with the highest throughput. At the same time, the mesh topology is, in every case, able to provide at least as much throughput as the best AP topology. In many cases, the mesh topology greatly surpasses the best AP topology, simply by allowing direct links to be used, without requiring that they be used in the cases where direct links provide low (or no) throughput.

VI. Related Work

Several recent studies have evaluated large wireless networks deployed across university campuses. Kotz and Essien [14] studied a 476 access point wireless network deployed across a large campus, focusing primarily on user traffic characteristics rather than link performance measurements. Other studies have investigated the characteristics of wireless links in sensor networks. Zhao and Govindan [10] measured the link characteristics of 60 sensor nodes deployed in an office building, an outdoor park, and a parking lot. The study finds that many links operate in a "gray area" with difficult-to-predict intermediate loss rates and performance.

The viability of multi-hop wireless mesh network topologies has been demonstrated for large outdoor and office networks in recent trial network and testbed deployments. Aguayo, et al. [1] deployed and maintained a 50-node community wireless mesh network used by students and faculty across a large university campus. In this deployment, each node in the network was able to communicate with other nodes and a small number of gateways to the Internet by using multi-hop routing and forwarding. Draves, et al. [6] used a 23-node wireless testbed deployed in an office environment to measure the performance of multi-hop mesh routing protocols and metrics. Their study demonstrates that selecting multi-hop routes that minimize end-to-end airtime based on link-level

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1151
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

measurements results in high-throughput performance in both single-radio and multi-radio mesh networks.

While it is not unexpected that wireless link performance will vary when deployed across large geographic areas, our study focuses specifically on the characteristics of home networks and demonstrates that variations in link quality are very common even when wireless networks are deployed within the relatively small area of a home. Moreover, this study demonstrates that the benefits of wireless mesh network topologies are not limited to wireless deployments spread across large campuses or office buildings. Mesh networking also improves the performance and reliability of small home wireless networks.

In a previous study, we provided early evidence of significant variability and asymmetry in home network link quality [11]. We have extended this early work in several important respects: (1) all new data sets, validating our previous results and claims, (2) measurement of throughput using TCP and UDP transports, (3) evaluation of the impact of automatic rate selection, and (4) comparison of the impact of flexible topologies on the performance of home wireless networks.

VII. Conclusions

Using six-node testbeds deployed in three different houses in the United States and the United Kingdom we studied the properties of home wireless networks. We showed that despite a home's relatively small size, the home is not a benign networking environment, and omnipresent connectivity is not guaranteed. Homes tend to feature wireless paths with a variety of obstacles which may render wireless communication impossible between node pairs.

Our results demonstrate that while wireless links inside homes tend to be stable over time there is significant performance variation across links, and many links are highly asymmetric. In home environments, precise node location is perhaps the single most important factor determining the quality of wireless communication. Indeed our results clearly confirm that distance has no impact on the quality of wireless links in the home, while small changes in antenna orientation and node location can dramatically change the performance of an individual link.

We have shown that the configuration parameters at the user's disposal today, power control, physical layer selection, and link rate, are not effective in achieving high-performance connectivity throughout a home. We found little difference between network performance at a moderate transmit power and the maximum transmit power. Links in our IEEE 802.11a and 802.11b networks had similar overall link characteristics, although the performance of 802.11a appears to be slightly better in the home. In both 802.11a and 802.11b networks, operating at the highest possible data rate tended to eliminate connectivity on a subset of links. Thus, the user is forced to either choose a lower data rate, sacrificing performance for connectivity, or use an auto rate selection algorithm. Unfortunately, auto rate selection may also fail to achieve the highest possible data rate. Moreover, auto rate selection cannot help if connectivity is poor at all rates.

While not typically controllable by a home user, we found that topology had the largest impact on overall network performance in the home. Using throughput measurements collected from all three homes, we have shown that the location of the access point can have a dramatic effect on the performance of a wireless network. In many cases, a given AP deployment will not yield a connected network. Since AP deployment is typically determined by the point of entry of the Internet service and aesthetic concerns, more flexible topologies may be more appropriate.

We considered two other types of topologies (e.g. direct and mesh) and found that mesh topologies offered significant benefits in the home. By selecting the topology according to measured link characteristics, such as loss rate, a mesh can provide more uniform connectivity while also allowing high-performance direct links where available. Used alone or in combination with the other wireless configuration parameters, mesh topologies can increase the performance of wireless networks in the home. These results suggest a need for support of mesh networking capabilities in wireless-enabled consumer electronic devices.

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11 mesh network," in *Proc. of ACM SIGCOMM*, Portland, OR, Sept. 2004.

[2] W. Rothman and T. Bradley, "Wi-fi versus your walls," *This Old House Magazine*, vol. 90, pp. 71–73, July/August 1995.

[3] IEEE 802.11 Task Group e, "Draft supplement to standard for telecommunications and information exchange between systems–lan/man specific requirements–part 11: Wireless medium access control (mac) and physical layer (phy) specification. medium access control (mac) quality of service (qos) enhancements," *IEEE std 802.11e draft 13.0*, Jan. 2005.

[4] S. J. Fortune, D. M. Gay, B. W. Kernighan, O. Landron, R. A. Valenzuela, and M. H. Wright, "Wise - a wireless system engineering tool," *IEEE Computational Science and Engineering*, vol. 2, no. 1, pp. 58–68, Mar. 1995.

[5] A. Verstak, J. He, L. T. Watson, N. Ramakrishnan, and C. A. Shaffer, "S4w: Globally optimized design of wireless communication systems," in *Proc. of Parallel and Distributed Processing Symposium*, Ft. Lauderdale, FL, Apr. 2002.

[6] R. Draves, J. Padhye, and B. Zill, "Comparison of routing metrics for static multi-hop wireless networks," in *Proc. of ACM SIGCOMM*, Portland, OR, Sept. 2004.

[7] IEEE 802.11 Task Group s, "Ieee 802.11 ess mesh networking par (802.11s)," May 2004. [Online]. Available: http://standards.ieee.org/board/nes/projects/802-11s.pdf

[8] J. Malinen, host AP driver for Intersil Prism2/2.5/3, http://hostap.epitest.fi/.

[9] J. Bicket, madwifi Stripped / Click Wifi Driver, http://www.pdos.lcs.mit.edu/~jbicket/madwifi.stripped/.

[10] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proc. of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, Nov. 2003.

[11] M. Yarvis, K. Papagiannaki, and W. Conner, "Characterization of 802.11 wireless networks in the home," in *Proc. of 1st Workshop on Wireless Network Measurements (WiNMee)*, Riva del Garda, Italy, 2005.

[12] K. Papagiannaki, M. Yarvis, and W. Conner, http://www.cambridge.intel-research.net/~kpapagia/infocom06_techreport.html.

[13] S. Lee, S. Banerjee, and B. Bhattacharjee, "The case for a multi-hop wireless local area network," in *Proc. of Conference of the IEEE Communications Society (INFOCOM)*, Hong Kong, Mar. 2004.

[14] D. Kotz and K. Essien, "Analysis of a campus-wide wireless network," in *Proc. of International Conference on Mobile Computing and Networking*, Atlanta, GA, Sept. 2002.

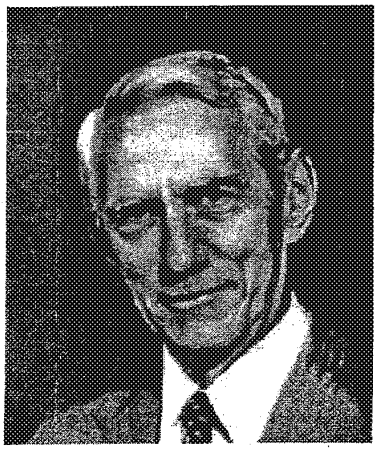
01000100
01100011 Data-Compression.com
A website devoted to the principles and practice of data compressio

Theory of Data Compression

Contents

I. Introduction and Background
II. Source Modeling
III. Entropy Rate of a Source
IV. Shannon Lossless Source Coding Theorem
V. Rate-Distortion Theory
VI. The Gap Between Theory and Practice
VII. FAQs (Frequently Asked Questions)
VIII. References

I. Introduction and Background



Claude E. Shannon

In his 1948 paper, ``A Mathematical Theory of Communication," Claude E. Shannon formulated the theory of data compression. Shannon established that there is a fundamental limit to lossless data compression. This limit, called the entropy rate, is denoted by H. The exact value of H depends on the information source --- more specifically, the statistical nature of the source. It is possible to compress the source, in a lossless manner, with compression rate close to H. It is mathematically impossible to do better than H.

Shannon also developed the theory of lossy data compression. This is better known as rate-distortion theory. In lossy data compression, the decompressed data does not have to be exactly the same as the original data. Instead, some amount of distortion, D, is tolerated. Shannon showed that, for a given source (with all its statistical properties known)

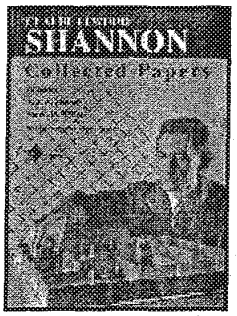
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1153
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

and a given distortion measure, there is a function, $R(D)$, called the rate-distortion function. The theory says that if D is the tolerable amount of distortion, then $R(D)$ is the best possible compression rate.

When the compression is lossless (i.e., no distortion or $D=0$), the best possible compression rate is $R(0)=H$ (for a finite alphabet source). In other words, the best possible lossless compression rate is the entropy rate. In this sense, rate-distortion theory is a generalization of lossless data compression theory, where we went from no distortion ($D=0$) to some distortion ($D>0$).

Lossless data compression theory and rate-distortion theory are known collectively as source coding theory. Source coding theory sets fundamental limits on the performance of all data compression algorithms. The theory, in itself, does not specify exactly how to design and implement these algorithms. It does, however, provide some hints and guidelines on how to achieve optimal performance. In the following sections, we will described how Shannon modeled an information source in terms of a random process, we will present Shannon lossless source coding theorem, and we will discuss Shannon rate-distortion theory. A background in probability theory is recommended.

II. Source Modeling



Available at Amazon.com

Imagine that you go to the library. This library has a large selection of books --- say there are 100 million books in this library. Each book in this library is very thick --- say, for example, that each book has 100 million characters (or letters) in them. When you get to this library, you will, in some random manner, select a book to check out. This chosen book is the information source to be compressed. The compressed book is then stored on your zip disk to take home, or transmitted directly over the internet into your home, or whatever the case may be.

Mathematically, the book you select is denoted by

$$\mathcal{X} = (X_1, X_2, X_3, X_4, \ldots).$$

where $\mathcal{X}$ represents the whole book, X_1 represents the first character in the book, X_2 represents the second character, and so on. Even though in reality the length of the book is finite, mathematically we assume that it has infinite length. The reasoning is that the book is so long we can just imagine that it goes on forever. Furthermore, the mathematics turn out to be surprisingly simpler if we assume an infinite length book. To simplify things a little, let us assume that all the characters in all the books are either a lower-case letter (`a' through `z') or a SPACE (e. e. cummings style of writing shall we say). The source alphabet, $\mathcal{A}$, is defined to be the set of all 27 possible values of the characters:

$$\mathcal{A} = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, SPACE\}.$$

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1154
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Now put yourself in the shoes of the engineer who designs the compression algorithm. She does not know in advance which book you will select. All she knows is that you will be selecting a book from this library. From her perspective, the characters in the book $(X_i,\ i = 1, 2, \ldots)$ are random variables which take values on the alphabet $\mathcal{A}$. The whole book, $\mathcal{X}$ is just an infinite sequence of random variables -- that is $\mathcal{X}$ is a random process. There are several ways in which this engineer can model the statistical properties of the book.

A. Zero-Order Model: Each character is statistically independent of all other characters and the 27 possible values in the alphabet $\mathcal{A}$ are equally likely to occur. If this model is accurate, then a typical opening of a book would look like this (all of these examples came directly from Shannon's 1948 paper):

xfoml rxkhrjffjuj zlpwcfwkcyj ffjeyvkcqsghyd qpaamkbzaacibzlhjqd

This does not look like the writing of an intelligent being. In fact, it resembles the writing of a ``monkey sitting at a typewriter."

B. First-Order Model: We know that in the English language some letters occur more frequently than others. For example, the letters `a' and `e' are more common than `q' and `z'. Thus, in this model, the character are still independent of one another, but the probability distribution of the characters are according to the first-order statistical distribution of English text. A typical text for this model looks like this:

ocroh hli rgwr nmielwis eu ll nbnesebya th eei alhenhttpa oobttva nah brl

C. Second-Order Model: The previous two models assumed statistical independence from one character to the next. This does not accurately reflect the nature of the English language. For exa#ple, some letters in thi# sent#nce are missi#g. However, we are still able to figure out what those letters should have been by looking at the context. This implies that there are some dependency between the characters. Naturally, characters which are in close proximity are more dependent than those that are far from each other. In this model, the present character X_i depends on the previous character X_{i-1} but it is conditionally independent of all previous characters $(X_1, X_2, \ldots, X_{i-2})$. According to this model, the probability distribution of the character X_i varies according to what the previous character X_{i-1} is.

For example, the letter `u' rarely occurs (probability=0.022). However, given that the previous character is `q', the probability of a `u' in the present character is much higher (probability=0.995). For a complete description, see the second-order statistical distribution of English text. A typical text for this model would look like this:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1155
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

on ie antsoutinys are t inctore st be s deamy achin d ilonasive tucoowe at teasonare fuso tizin andy tobe seace ctisbe

D. Third-Order Model: This is an extension of the previous model. Here, the present character X_i depends on the previous two characters (X_{i-2}, X_{i-1}) but it is conditionally independent of all previous characters before those: $(X_1, X_2, \ldots, X_{i-3})$. In this model, the distribution of X_i varies according to what (X_{i-2}, X_{i-1}) are. See the third-order statistical distribution of English text. A typical text for this model would look like this:

in no ist lat whey cratict froure birs grocid pondenome of demonstures of the reptagin is regoactiona of cre

The resemblance to ordinary English text increases quite noticeably at each of the above steps.

E. General Model: In this model, the book $\mathcal{X}$ is an arbitrary stationary random process. The statistical properties of this model are too complex to be deemed practical. This model is interesting only from a theoretical point of view.

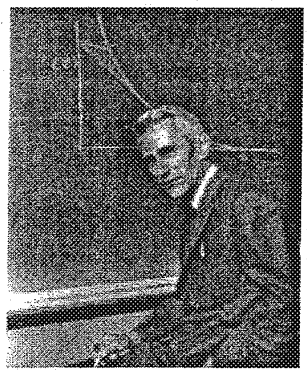
Model A above is a special case of Model B. Model B is a special case of Model C. Model C is a special case of Model D. Model D is a special case of Model E.

III. Entropy Rate of a Source

The entropy rate of a source is a number which depends only on the statistical nature of the source. If the source has a simple model, then this number can be easily calculated. Here, we consider an arbitrary source:

$$\mathcal{X} = (X_1, X_2, X_3, X_4, \ldots),$$

while paying special attention to the case where $\mathcal{X}$ is English text.



Shannon in 1948

A. Zero-Order Model: The characters are statistically independent of each other and every letter of the alphabet, $\mathcal{A}$, are equally likely to occur. Let m be the size of the alphabet. In this case, the entropy rate is given by

$$H = \log_2 m \quad \text{bits/character}$$

For English text, the alphabet size is $m=27$. Thus, if this had been an accurate

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1156
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

model for English text, then the entropy rate would have been $H=\log_2 27=4.75$ bits/character.

B. First-Order Model: The characters are statistically independent. Let m be the size of the alphabet and let p_i be the probability of the i-th letter in the alphabet. The entropy rate is

$$H = -\sum_{i=1}^{m} p_i \log_2 p_i \quad \text{bits/character.}$$

Using the first-order distribution, the entropy rate of English text would have been 4.07 bits/character had this been the correct model.

C. Second-Order Model: Let $P_{j|i}$ be the conditional probability that the present character is the j-th letter in the alphabet given that the previous character is the i-th letter. The entropy rate is

$$H = -\sum_{i=1}^{m} p_i \sum_{j=1}^{m} P_{j|i} \log_2 P_{j|i} \quad \text{bits/character.}$$

Using the second-order distribution, the entropy rate of English text would have been 3.36 bits/character had this been the correct model.

D. Third-Order Model: Let $P_{k|j,i}$ be the conditional probability that the present character is the k-th letter in the alphabet given that the previous character is the j-th letter and the one before that is the i-th letter. The entropy rate is

$$H = -\sum_{i=1}^{m} p_i \sum_{j=1}^{m} P_{j|i} \sum_{k=1}^{m} P_{k|j,i} \log_2 P_{k|j,i} \quad \text{bits/character.}$$

Using the third-order distribution, the entropy rate of English text would have been 2.77 bits/character had this been the correct model.

E. General Model: Let B_n represents the first n characters. The entropy rate in the general case is given by

$$H = \lim_{n\to\infty} -\frac{1}{n} \sum p(B_n) \log_2 p(B_n) \quad \text{bits/character,}$$

where the sum is over all m^n possible values of B_n. It is virtually impossible to calculate the entropy rate according to the above equation. Using a prediction method, Shannon has been able to estimate that the entropy rate of the 27-letter English text is 2.3 bits/character (see Shannon:Collected Papers).

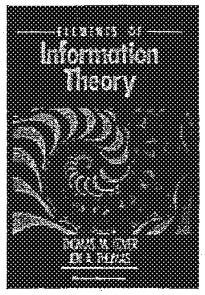
We emphasize that there is only one entropy rate for a given source. All of the above definitions for the entropy rate are consistent with one another.

IV. Shannon Lossless Source Coding Theorem

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1157
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Shannon lossless source coding theorem is based on the concept of block coding. To illustrate this concept, we introduce a special information source in which the alphabet consists of only two letters:

$$\mathcal{A} = \{a, b\}.$$



Excellent Textbook

Here, the letters `a' and `b' are equally likely to occur. However, given that `a' occurred in the previous character, the probability that `a' occurs again in the present character is 0.9. Similarly, given that `b' occurred in the previous character, the probability that `b' occurs again in the present character is 0.9. This is known as a binary symmetric Markov source.

An n-th order block code is just a mapping which assigns to each block of n consecutive characters a sequence of bits of varying length. The following examples illustrate this concept.

1. First-Order Block Code: Each character is mapped to a single bit.

B_1	$p(B_1)$	Codeword
a	0.5	0
b	0.5	1
R=1 bit/character		

An example:

Original Data: a a a a a a a b b b b b b b b b b b b b a a a a
Compressed Data: 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0

Note that 24 bits are used to represent 24 characters --- an average of 1 bit/character.

2. Second-Order Block Code: Pairs of characters are mapped to either one, two, or three bits.

B_2	$p(B_2)$	Codeword
aa	0.45	0
bb	0.45	10
ab	0.05	110
ba	0.05	111
R=0.825 bits/character		

An example:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1158
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Original Data:	a a	a a	a a	a b	b b	b b	b b	b b	b b	b b	b b	a a	a a
Compressed Data:	0	0	0	110	10	10	10	10	10	10	0	0	

Note that 20 bits are used to represent 24 characters --- an average of 0.83 bits/character.

3. Third-Order Block Code: Triplets of characters are mapped to bit sequence of lengths one through six.

B_3	$p(B_3)$	Codeword
aaa	0.405	0
bbb	0.405	10
aab	0.045	1100
abb	0.045	1101
bba	0.045	1110
baa	0.045	11110
aba	0.005	111110
bab	0.005	111111
R=0.68 bits/character		

An example:

Original Data:	a a a	a a a	a b b	b b b	b b b	b b b	b b a	a a a
Compressed Data:	0	0	1101	10	10	10	1110	0

Note that 17 bits are used to represent 24 characters --- an average of 0.71 bits/character.

Note that:

- The rates shown in the tables are calculated from

$$R = \frac{1}{n}\sum p(B_n)l(B_n) \quad \text{bits/sample,}$$

where $l(B_n)$ is the length of the codeword for block B_n.

- The higher the order, the lower the rate (better compression).
- The codes used in the above examples are Huffman codes.
- We are only interested in lossless data compression code. That is, given the code table and given the compressed data, we should be able to rederive the original data. All of the examples given above are lossless.

Theorem: *Let R_n^* be the rate of an optimal n-th order lossless data compression code (in bits/character). Then*

$$-\frac{1}{n}\sum p(B_n)\log_2 p(B_n) \le R_n^* < -\frac{1}{n}\sum p(B_n)\log_2 p(B_n) + \frac{1}{n}.$$

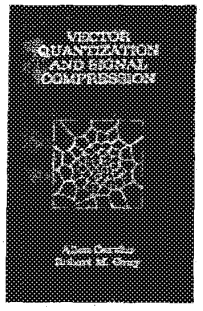
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1159
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Since both upper and lower bounds of R_n^* approach the entropy rate, H, as n goes to infinity, we have

$$\lim_{n\to\infty} R_n^* = H.$$

Thus, the theorem established that the entropy rate is the rate of an optimal lossless data compression code. The limit exists as long as the source is stationary.

V. Rate-Distortion Theory



Available at Amazon.com

In lossy data compression, the decompressed data need not be exactly the same as the original data. Often, it suffices to have a reasonably close approximation. A distortion measure is a mathematical entity which specifies exactly how close the approximation is. Generally, it is a function which assigns to any two letters x and $\hat{x}$ in the alphabet $\mathcal{A}$ a non-negative number denoted as

$$d(x, \hat{x}) \geq 0.$$

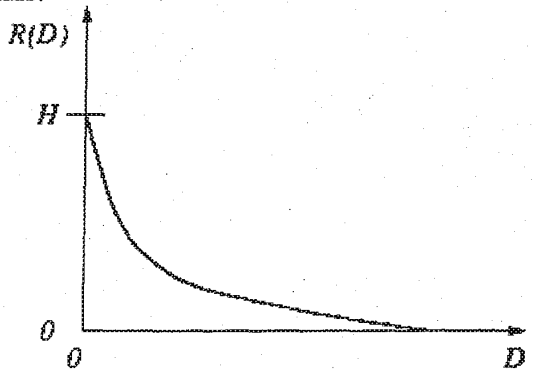
Here, x is the original data, $\hat{x}$ is the approximation, and $d(x, \hat{x})$ is the amount of distortion between x and $\hat{x}$. The most common distortion measures are the Hamming distortion measure:

$$d(x, \hat{x}) = \begin{cases} 0 & \text{if } x = \hat{x} \\ 1 & \text{if } x \neq \hat{x} \end{cases},$$

and the squared-error distortion measure (which is only applicable when $\mathcal{A}$ is a set of numbers):

$$d(x, \hat{x}) = (x - \hat{x})^2.$$

Rate-distortion theory says that for a given source and a given distortion measure, there exists a function, *R(D)*, called the rate-distortion function. The typical shape of *R(D)* looks like this:



If the source samples are independent of one another, the rate-distortion function can be obtained by solving the constrained minimization problem:

$$R(D) = \min_{Q_{j|i}} \sum_{i=1}^{m} p_i \sum_{j=1}^{m} Q_{j|i} \log_2 \left[\frac{Q_{j|i}}{\sum_{k=1}^{m} p_k Q_{j|k}} \right],$$

subject to the constraints that $0 \le Q_{j|i} \le 1, \sum_{j=1}^{m} Q_{j|i} = 1$ and

$$\sum_{i=1}^{m} p_i \sum_{j=1}^{m} Q_{j|i} d(i,j) \le D,$$

where $d(i,j)$ is the distortion between the i-th and j-th letter in the alphabet. Using Blahut's algorithm, it may be possible to numerically calculate the rate-distortion function.

Rate-distortion theory is based on the concept of block coding (similar to above). A lossy block code is known as a vector quantizer (VQ). The block length n of the code is known as the VQ dimension.

> **Theorem:** *Assuming that the source is stationary and the source samples are independent. For every $D \ge 0$ and for every $\epsilon \ge 0$, there exists a VQ of (sufficiently large) dimension n with distortion no greater than*
>
> $$D + \epsilon,$$
>
> *and rate no more than*
>
> $$R(D) + \epsilon.$$
>
> *Further more, there does not exist a VQ with distortion D and rate less than $R(D)$.*

In essence, the theorem states that *R(D)* is the rate-distortion performance of an optimal VQ. The above theorem can be generalized to the case where the source is stationary and ergodic.

VI. The Gap Between Theory and Practice

- The theory holds when the block length n approaches infinity. In real-time compression, the compression algorithm must wait for n consecutive source samples before it can begin the compression. When n is large, this wait (or delay) may be too long. For example, in real-time speech compression, the speech signal is sampled at 8000 samples/second. If n is say 4000, then the compression delay is half a second. In a two-way conversation, this long delay may not be desirable.
- The theory does not take into consideration the complexities associated with the compression and decompression operations. Typically, as the block length n increases, the complexities also increase. Often, the rate of increase is

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1161
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

exponential in n.

- The theory assumes that the statistical properties of the source is known. In practice, this information may not be available.
- The theory assumes that there is no error in the compressed bit stream. In practice, due to noise in the communication channel or imperfection in the storage medium, there are errors in the compressed bit stream.

All of these problems have been successfully solved by researchers in the field.

VII. FAQs (Frequently Asked Questions)

A. What is the difference between lossless and lossy compression?
B. What is the difference between compression rate and compression ratio?
C. What is the difference between ``data compression theory" and ``source coding theory"?
D. What does ``stationary" mean?
E. What does ``ergodic" mean?

A. What is the difference between lossless and lossy compression?

In lossless data compression, the compressed-then-decompressed data is an exact replication of the original data. On the other hand, in lossy data compression, the decompressed data may be different from the original data. Typically, there is some distortion between the original and reproduced signal.

The popular WinZip program is an example of lossless compression. JPEG is an example of lossy compression.

B. What is the difference between compression rate and compression ratio?

Historically, there are two main types of applications of data compression: transmission and storage. An example of the former is speech compression for real-time transmission over digital cellular networks. An example of the latter is file compression (e.g. Drivespace).

The term ``compression rate" comes from the transmission camp, while ``compression ratio" comes from the storage camp.

Compression rate is the rate of the compressed data (which we imagined to be transmitted in ``real-time"). Typically, it is in units of bits/sample, bits/character, bits/pixels, or bits/second. Compression ratio is the ratio of the size or rate of the original data to the size or rate of the compressed data. For example, if a gray-scale image is originally represented by 8 bits/pixel (bpp) and it is compressed to 2 bpp, we say that the compression ratio is 4-to-1. Sometimes, it is said that the compression ratio is 75%.

Compression rate is an absolute term, while compression ratio is a relative term.

We note that there are current applications which can be considered as both transmission and storage. For example, the above photograph of Shannon is stored in JPEG format. This not only saves storage space on the local disk, it also speeds up the delivery of the image over the internet.

C. **What is the difference between ``data compression theory'' and ``source coding theory''?**

There is no difference. They both mean the same thing. The term ``coding'' is a general term which could mean either ``data compression'' or ``error control coding''.

D. **What does ``stationary'' mean?**

Mathematically, a random process is stationary (or strict-sense stationary) if for every positive integers n and k the vectors:

$$(X_1, X_2, \ldots, X_n)$$

and

$$(X_{1+k}, X_{2+k}, \ldots, X_{n+k})$$

have the same probability distribution.

We can think of ``stationarity'' in terms of our library example. Suppose that we look at the first letter of all 100 million books and see how often the first letter is `a', how often it is `b', and so on. We will then get a probability distribution of the first letter of the (random) book. If we repeat this experiment for the fifth and 105th letters, we will get two other distributions. If all three distributions are the same, we are inclined to believe that the book process is stationary. To be sure, we should look at the joint distribution of the first and second letters and compare it with the joint distribution of the 101st and 102nd letters. If these two joint distributions are roughly the same, we will be more convinced that the book process is stationary.

In reality, the book process can not be stationary because the first character can not be a SPACE.

E. **What does ``ergodic'' mean?**

The strict mathematical definition of ergodicity is too complex to explain. However, Shannon offered the following intuitive explanation: ``In an ergodic process every sequence produced by the process is the same in statistical properties. Thus the letter frequencies, (the pairwise) frequencies, etc., obtained from particular sequences, will, as the lengths of the sequences increase, approach definite limits independent of the particular sequence. Actually this is not true of every sequence but the set for which it is false has probability zero. Roughly the ergodic property means statistical homogeneity."

VIII. References

1. C. E. Shannon, A Mathematical Theory of Communication (free pdf version)
2. C. E. Shannon and W. Weaver, The Mathematical Theory of Communication. ($13 paper version)
3. C. E. Shannon, N. J. Sloane, and A. D. Wyner, Claude Elwood Shannon: Collected Papers.
4. C. E. Shannon, ``Prediction and Entropy of Printed English," available in Shannon: Collected Papers.
5. C. E. Shannon, ``Coding Theorems for a Discrete Source with a Fidelity Criterion," available in Shannon: Collected Papers.
6. T. M. Cover and J. A. Thomas, Elements of Information Theory.
7. R. Gallager, Information Theory and Reliable Communication.
8. A. Gersho and R. M. Gray, Vector Quantization and Signal Compression.
9. K. Sayood, Introduction to Data Compression.
10. M. Nelson and J.-L. Gailly, The Data Compression Book.
11. A. Leon-Garcia, Probability and Random Processes for Electrical Engineering. (Student Solution Manual).
12. A. Papoulis, Probability, Random Variables, and Stochastic Processes.
13. M. R. Schroeder, Computer Speech: Recognition, Compression, Synthesis.
14. G. Held and T. R. Marshall, Data and Image Compression: Tools and Techniques.
15. D. Hankerson, P. D. Johnson, and G. A. Harris, Introduction to Information Theory and Data Compression .

Note: To anyone interested in information theory, I can highly recommend two books: Shannon's Collected Papers and Cover and Thomas' Elements of Information Theory. Collected Papers includes a 1987 interview of Shannon by OMNI Magazine, almost all of his papers on information theory and cryptography, his celebrated Master's Theses on *A Symbolic Analysis of Relay and Switching Circuits*, his Ph.D. dissertation on *An Algebra for Theoretical Genetics*, his paper on the *Scientific Aspects of Juggling*, his paper on *A Mind Reading Machine*, and much more. Elements of Information Theory is now the standard textbook on information theory. I have been using this textbook in my graduate-level information theory course for the past seven years. I have had nothing but positive feedback from students who have studied this book.

Nam Phamdo
Department of Electrical and Computer Engineering
State University of New York
Stony Brook, NY 11794-2350
phamdo@ieee.org

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1164
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)



RGB Networks

Comparing Adaptive HTTP Streaming Technologies

A Comparison of Apple's HTTP Live Streaming (HLS), Microsoft's Silverlight Smooth Streaming (MSS) and Adobe's HTTP Dynamic Streaming (HDS)

1. Introduction

Watching video online or on the go on a tablet or mobile phone is no longer a novelty, nor is streaming Internet-delivered content to the TV in your living room. Driven by the boom in video-enabled devices, including PCs, smartphones, and Internet-enabled set-top boxes and televisions, consumers have rapidly moved through the early-adopter phase of TV Everywhere to the stage where a growing mass of consumers expect that any media should be available on any device over any network connection, delivered at the same high quality they've come to expect from traditional television services. This explosion of multiscreen IP video – whether regarded as disruption for traditional pay-TV providers or an opportunity to expand their services – is definitely here to stay.

While tremendous advancements in core and last mile bandwidth have been achieved in the last decade – primarily driven by Web-based data consumption – video traffic represents a leap in bandwidth requirements. This, coupled with the fact that the Internet at large is not a managed quality-of-service (QoS) environment, requires that new methods of video transport be considered to provide the same quality of video experience across any device and network that consumers have come to expect from managed TV delivery networks.

The evolution of video delivery transport has led to a new set of de facto standard adaptive delivery protocols from Apple, Microsoft and Adobe that are now positioned for broad adoption. Consequently, networks must now be equipped with servers that can take high-quality video content from live streams or file sources and 'package' it for transport to devices ready to accept these new delivery protocols.

This paper gives a technical comparison of the three main HTTP adaptive streaming technologies available today: Apple's HTTP Live Streaming (HLS), Microsoft Silverlight Smooth Streaming (MSS) and Adobe's HTTP Dynamic Streaming (HDS), also previously referred to as 'Zeri.' The paper is divided into three main parts: (1) it begins by giving an overview of adaptive HTTP streaming, discussing delivery architectures, highlighting its strengths and weaknesses, and discussing live and video-on-demand (VoD) delivery; (2) it then delves into each technology, explaining how they work and highlighting how each technology is different from the others; (3) finally, it looks at specific features and describes how they are implemented or deployed. This last section focuses on:

- Delivery of multiple audio channels
- Encryption and DRM
- Closed captions / subtitling
- Ability to insert ads
- Custom VOD playlists
- Trick modes (fast-forward/rewind, pause)
- Fast channel change

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1166
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Failover due to upstream issues
- Stream latency
- Ability to send other data to the client, including manifest compression

2. Adaptive HTTP Video Delivery

Background

In 'traditional' IP streaming, a video server sends video to a client at a fixed bandwidth. The client and server must have synchronized states: e.g. 'video is stopped,' 'video is playing,' 'video is paused,' etc. Traditional video streaming is typically delivered over UDP, a connectionless protocol in which packet losses result in poor quality-of-experience (QoE) and which has difficulty passing through firewalls in home routers. Traditional streaming can adapt to changes in network bandwidth, but only with complex synchronization between the server and client, and as a result, such adaptive protocols were never widely adopted.

In comparison, in adaptive HTTP streaming the source video, whether a file or a live stream, is encoded into file segments – sometimes referred to as 'chunks' or 'segments' – using a desired format, which includes a container, video codec, audio codec, encryption protocol, etc. Segments typically represent two to ten seconds of video. The stream is broken into segments at video Group of Pictures (GOP) boundaries that begin with an IDR frame (a frame that can be independently decoded without dependencies on other frames), giving each segment independence from previous and successive segments. The segments are subsequently hosted on a regular HTTP server. Each sequence of segments is called a profile. Profiles may differ in bitrate, resolution, codec or codec profile/level.

Clients play the stream by requesting segments in a profile from a Web server, downloading them via HTTP. As the segments are downloaded, the client plays back the segments in the order requested. Since the segments are sliced along GOP boundaries with no gaps between, video playback is seamless – even though it is actually just a collection of independent file downloads via a sequence of HTTP GET requests.

Adaptive delivery enables a client to 'adapt' to fluctuating network conditions by selecting video segments from different profiles. The client can easily compute the available network bandwidth by comparing the download time of a segment with its size. If the client has a list of available profile bitrates (or resolutions or codecs), it can determine if it must change to a lower bitrate/resolution profile or whether the available bandwidth allows it to download segments from a higher bitrate/resolution profile. This list of available profiles is called a manifest or playlist. The client's bandwidth calculation is repeated at every chunk download, and so the client can adapt to changing network bandwidth or other conditions every few seconds. Aside from network bandwidth, conditions that may affect the client's choice of profile may include local CPU load or the client's ability to play back a specific codec or resolution.

This delivery model works for both live- and file-based sources. In either case, a manifest file is

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1167
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

provided to the client. The manifest lists the bitrates (and potentially other data) associated with the available stream profiles and the client uses it to determine how to download the chunks; that is, what URL to use to fetch chunks from specific profiles. In the case of an on-demand file request, the manifest contains information on every chunk in the content. In the case of live streaming, this isn't possible. HLS and HDS deliver a 'rolling window' manifest data that contains references to the last few available chunks, as shown in Figure 1. The client must update its manifest repeatedly in order to know about the most recently available chunks. MSS delivers information in each chunk that lets the client access subsequent chunks, so no rolling window-type manifest is needed.

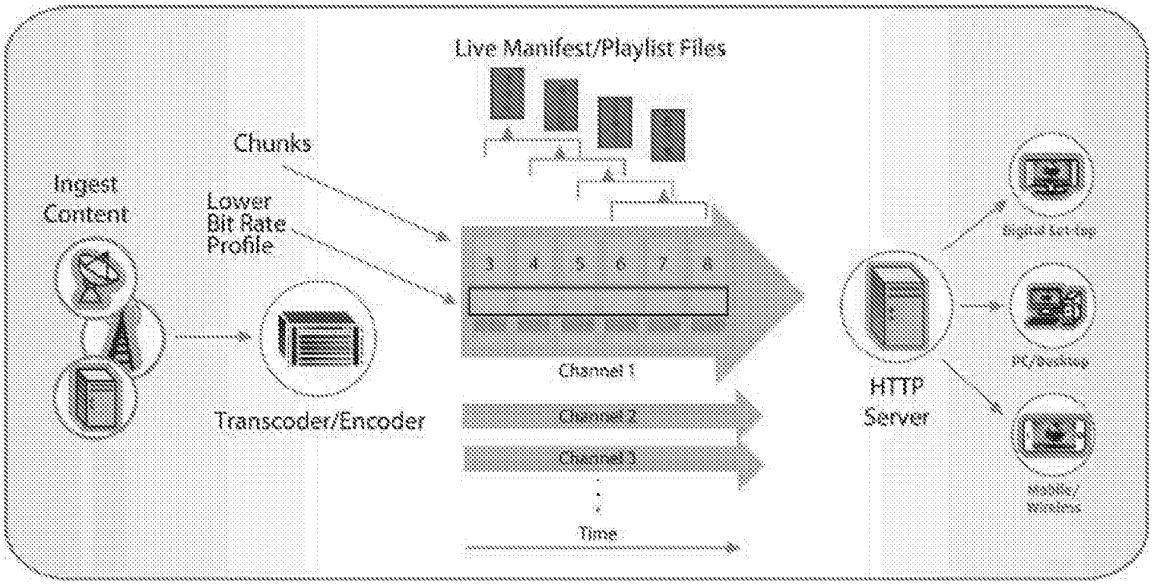


Figure 1. The content delivery chain for a live adaptive HTTP stream. The client downloads the 'rolling window' manifest files that refer to the latest available chunks. The client then uses these references to download chunks and play them back sequentially. In the figure, the first manifest refers to chunks 3, 4, and 5, which are available in multiple bitrates. As new chunks become available the playlist is updated to reference the latest available chunks.

Adaptive HTTP streaming has a number of advantages over traditional streaming:

- Lower infrastructure costs for content providers by eliminating specialty streaming servers in lieu of generic HTTP caches/proxies (that may already be in place for HTTP data delivery);
- Content delivery is dynamically adapted to the weakest link in the end-to-end delivery chain, including highly varying last mile conditions;
- Subscribers no longer need to statically select a bitrate on their own, as the client can now perform that function dynamically and automatically;
- Subscribers enjoy fast start-up and seek times as playback control functions can be initiated via the lowest bitrate chunks and subsequently ratcheted up to a higher bitrates;
- Annoying user experience shortcomings, including long initial buffer time, disconnects and playback start/stop are virtually eliminated;

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1168
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- The client can control bitrate switching – with no intelligence in the server – taking into account CPU load, available bandwidth, resolution, codec and other local conditions;
- HTTP delivery works through firewalls and NAT.

Adaptive HTTP streaming also has some consequences:

- The clients must buffer a few chunks to make sure they don't starve their input buffers, which increases the end-to-end latency of live streams;
- HTTP is based on TCP; when packet loss is low, TCP recovers well and this means that video playback has no artifacts caused by missing data. However, when packet loss rises, TCP can fail completely. Thus, overall, clients will typically have good quality playback or playback that stops completely, as opposed to quality that degrades proportionally to the amount of packet loss in the delivery network. The internet is generally reliable enough so that the benefit of completely clean video at low packet drop rates (with TCP) supersedes the value of some, but very poor quality, video at high packet drop rates (with UDP).

2.1 Video Delivery Components

At a high level, the components in an adaptive HTTP streaming data flow consist of an encoder or transcoder, a packager (also called a segmenter or a fragmenter), and a content delivery network (CDN). This section discusses the features of these components (see Figure 2) that are related to adaptive streaming.

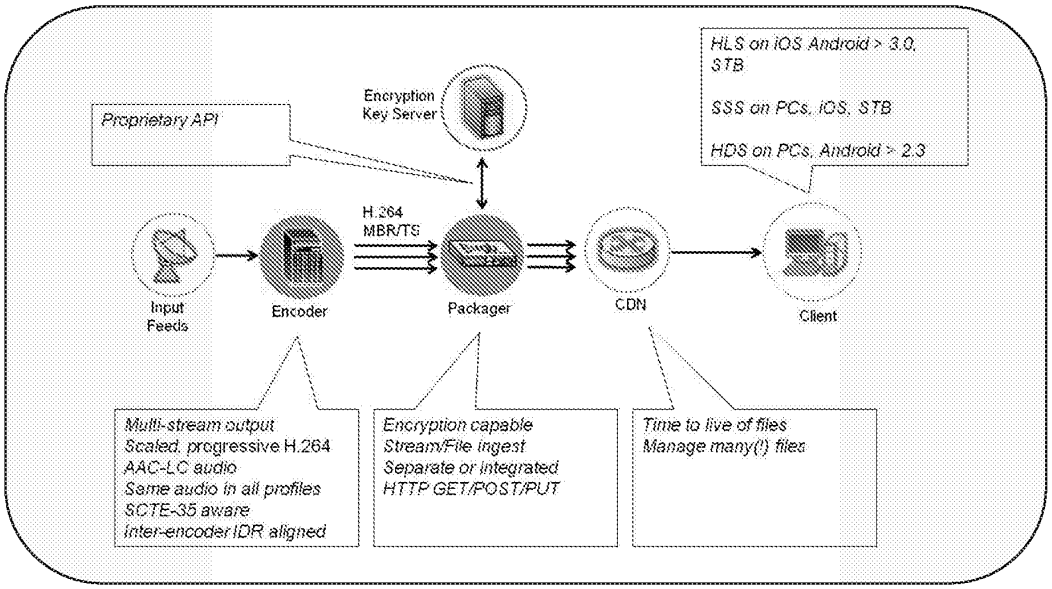


Figure 2. The components of an HTTP streaming system, with some highlighted features.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1169
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The Encoder/Transcoder

The transcoder (or encoder, if the input is not already encoded in another format) is responsible for ingesting the content and preparing it for segmentation. The transcoder must process the video in the following way:

- The output video must be in progressive format. The transcoder must therefore de-interlace the input.
- The output video must be scaled to resolutions suitable for the client device.
- The different output profiles must be IDR-aligned so that the client playback of the chunks created from each profile is continuous and smooth.
- Audio must be transcoded into AAC audio, the codec used by HLS, HDS, and MSS.
- The same encoded audio stream needs to be streamed on all the output video profiles; this avoids clicking artifacts during client-side profile changes.
- If SCTE 35 is used for ad insertion, it is desirable to have the transcoder add IDR frames at the ad insertion points so that the video is ready for ad insertion. It is then possible to align the chunk boundaries with the ad insertion points, so that ad insertion can be done more easily (via substitution of chunks) as compared to the traditional stream splicing.
- A desirable fault tolerance mechanism allows two different transcoders that ingest the same input to create identically IDR-aligned output. This can be used to create a redundant backup of encoded content in such a way that any failure of the primary transcoder is seamlessly backed up by the secondary transcoder.

Because the quality of experience of the client depends on having a number of different profiles, it is desirable to have an encoder that can output a large number of different profiles for each input. Deployments may use anywhere from 4 to 16 different output profiles for each input, with more profiles resulting in more supported devices and a better user experience. The table below shows a typical use case for the different output profiles:

Width	Height	Video Bitrate
1280	720	3 Mbits/s
960	540	1.5 MBits/s
864	486	1.25 MBits/s
640	360	1.0 MBits/s
640	360	750 Kbits/s
416	240	500 Kbits/s
320	180	350 Kbits/s
320	180	150 Kbits/s

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1170
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The Packager

The packager is the component that takes the output of the transcoder and packages the video for a specific delivery protocol. The packager should have the following features:

- Encryption capability – the packager should be able to encrypt the outgoing chunks in a format compatible with the delivery protocol.
- Integration with third party key management systems – the packager should be able to receive encryption keys from a third party key management server that is also used to manage and distribute the keys to the clients.
- Packagers may ingest live streams or files, depending on whether the workflow is live or on-demand.
- Packagers should support multiple ways to deliver the chunks, either via HTTP pull or by pushing the chunks via a network share or HTTP PUT / POST.

The CDN

The content delivery network (CDN) needed for HTTP streaming is not specialized – it is HTTP-based and doesn't require any special streaming servers. For live delivery, it is beneficial to tune the CDN to age out older chunks rapidly, as there is no need to keep them around long. The actual duration depends on the duration of the chunks and latency in the client, but a minute is normally sufficient.

It is important to note that the number of chunks can be very large. For example, a day's worth of 2-second chunks delivered in 10 different profiles for 100 different channels creates 43 million files! Thus the CDN must be able to cope with a large number of files as well.

The Client

HLS is available on iOS devices, but availability on Windows is via third party products that are not always complete or robust. Interestingly, however, Android 3.0 supports HLS natively, though some features aren't well supported (e.g. stream discontinuity indications). MSS on a PC depends on a Silverlight runtime client which must be installed. But there are native Smooth Streaming clients developed for multiple devices, including iOS tablets, phones and iPods. HDS is native to flash 10.1 and later, and comes with the Flash plug-in on PCs, as well as on Android 2.3 and later devices.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1171
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Workflow Architecture

It is valuable to have an architecture that allows the transcoder and packager to be separate. This has the advantage that the input video can be transcoded just once at the core, delivered over a core network to the edge, and packaged into multiple formats at the edge of the network. Without this separation, all the final delivered formats must be delivered over the core network, unnecessarily increasing its bandwidth utilization. This is shown in Figure 3. Note, however, that packet loss on the core network would result in unrecoverable segment loses.

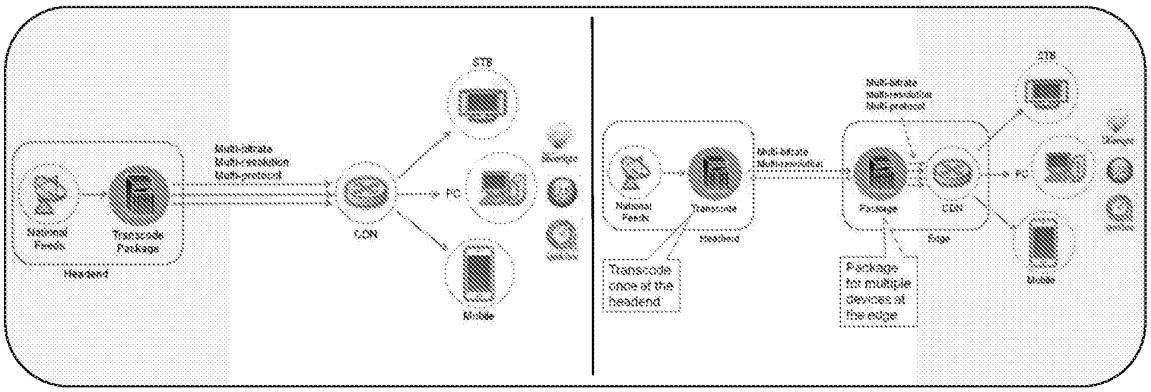


Figure 3. Integrated and remote segmentation of streams: When multiple formats are used, segmenting closer to the edge of the network (shown on the right) can save core bandwidth, as streams only need to be delivered once and can be packaged into multiple delivery formats at the edge. However, if the core network is susceptible to packet loss, segmenting at the core ensures that segments will always be delivered to the CDN (shown on the left).

Other Adaptive HTTP Streaming Technologies

There are a number of other adaptive streaming technologies available also, with varying market penetration:

- Move Networks went out of business and was purchased by Echostar – their team is now integrating the technology into their home devices. Move was instrumental in popularizing adaptive HTTP streaming and has a number of patents on the technology (though chunked streaming was used before Move popularized it).
- 3GPP's Adaptation HTTP Streaming (AHS) is part of 3GPPs rel 9 specification (see [AHS]). And 3GPP rel 10 is working on a specification called DASH as well.
- The Open TV Forum has an HTTP Adaptive Streaming (HAS) specification (see [HAS]).
- MPEG Dynamic Adaptive Streaming over HTTP (DASH) is based on 3GPP's AHS and the Open TV Forum's HAS and is close to completion. It specifies use of either fMP4 or transport stream (TS) chunks and an XML manifest (called the media presentation description or MPD) that is repeatedly downloaded. DASH may well become the format of choice in the future, but currently, the lack of client support makes this specification interesting only in theory. DASH

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1172
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

does make allowances for multiple scenarios, including separate or joined streaming of audio, video and data, as well as encryption. However, its generality is as much a drawback as an advantage, since it makes clients complex to implement.

- Many DRM vendors have their own variation of these schemes.

3. Adaptive Streaming Internals

In this section we review some of the details of HLS, HDS and MSS. Each of these protocols has strengths and weaknesses, which we discuss in the following sections.

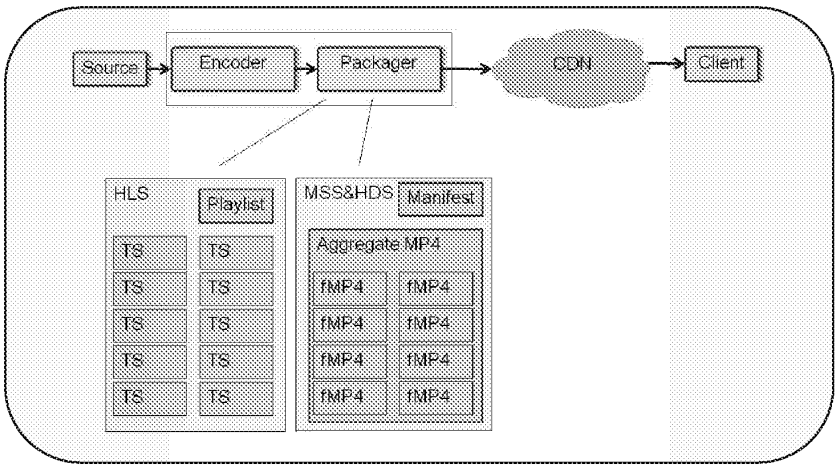


Figure 4. A comparison of HLS and MSS/HDS: The latter can create aggregate formats that can be distributed on a CDN for VoD, whereas for live video, all distribute chunks on the CDN. MSS allows audio and video to be aggregated and delivered separately, but HDS and HLS deliver these together.

Apple HTTP Live Streaming (HLS)

Ironically, unlike Microsoft and Adobe, Apple chose not to use the ISO MPEG file format – a format based on Apple's MOV file format – in its adaptive streaming technology. Instead, HLS takes an MPEG-2 TS and segments it to a sequence of MPEG-2 TS files which encapsulate both the audio and video. These segments are placed on any HTTP server along with the playlist files. The playlist (or index) manifest file is a text file (based on Winamp's original m3u file format) with an m3u8 extension. Full details can be found in [HLS].

HLS defines two types of playlist files: normal and variant. The normal playlist file lists URLs that point to chunks that should be played sequentially. The variant playlist files points to a collection of different normal playlist files, one for each output profile. Metadata is carried in the playlist files as comments – lines preceded by '#'. In the case of normal playlist files, this metadata includes a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1173
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

sequence number used to associate chunks from different profiles, information about the chunk duration, a directive signaling whether chunks can be cached, the location of decryption keys, the type of stream, and time information. In the case of a variant playlist the metadata includes the bitrate of the profile, its resolution, its codec, and an ID that can be used to associate different encodings of the same content. Figure 5 and Figure 6 show a sample HLS variant playlist file and normal playlist file.

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=531475
mystic_S1/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=381481
mystic_S2/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=531461
mystic_S3/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=781452
mystic_S4/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1031452
mystic_S5/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1281452
mystic_S6/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1531464
mystic_S7/mnf.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=3031464
mystic_S8/mnf.m3u8
```

Figure 5. An HLS variant playlist file showing eight output profiles with different bitrates. The URLs for the m3u8 files are relative, but could include the leading 'http://...'. In this example, each profile's playlist is in a separate path component of the URL.

```
#EXTM3U
#EXT-X-KEY:METHOD=NONE
#EXT-X-TARGETDURATION:11
#EXT-X-MEDIA-SEQUENCE:494

#EXT-X-KEY:METHOD=NONE
#EXTINF:10,505.ts
505.ts
#EXTINF:10,506.ts
506.ts
#EXTINF:10,507.ts
507.ts
```

Figure 6. An HLS playlist file from a live stream showing the three latest available TS chunks. The `#EXT-X-MEDIA-SEQUENCE` tag identifies the sequence number of the first chunk, 505.ts; it is used to align chunks from different profiles. Note that the chunk name carries no streaming-specific information. The `#EXT-X-TARGETDURATION:11` tag is the expected duration (10 seconds) of the chunks, though durations can vary. The `#EXT-X-KEY:METHOD=NONE` tag shows that no encryption was used in this sequence. The `#EXTINF:10` tags show the duration of each segment. As in the variant playlist file, the URLs are relative to the base URL used to fetch the playlist.

In HLS, a playlist file corresponding to a live stream must be repeatedly downloaded so that the client can know the URLs of the most recently available chunks. The playlist is downloaded every time a chunk is played, and thus, in order to minimize the number of these requests, Apple recommends a relatively long chunk duration of 10 seconds. However, the size of the playlist file is small compared with any video content, and the client maintains an open TCP connection to the server, so that this network load is not significant. Shorter chunk durations can thus be used. This allows the client to adapt bitrates more quickly. VoD playlists are distinguished from live playlists by the `#EXT-X-PLAYLIST-TYPE` and `#EXT-X-ENDLIST` tags.

HLS is the only protocol that doesn't require chunks to start with IDR frames. It can download chunks from two profiles and switch the decoder between profiles on an IDR frame that occurs in

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1174
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

the middle of a chunk. However, doing this incurs extra bandwidth consumption, as two chunks corresponding to the same portion of video are downloaded at the same time.

HLS Considerations
Some advantages of HLS are:

- It is a simple protocol that is easy to modify. The playlists are easily accessible and their text format lends to simple modification for applications such a re-broadcast or ad insertion.
- The use of TS files means that there is a rich ecosystem for testing and verifying file conformance.
- TS files can carry other metadata, such as SCTE 35 cues or ID3 tags (see [HLSID3]).
- HLS is native to popular iOS devices, the users of which are accustomed to paying for apps and other services. That is, HLS is more easily monetized.

Some disadvantages of HLS are:

- HLS is not supported natively on Windows OS platforms.
- TS files mux the audio, video and data together. This means that multi-language support either comes at the cost of sending all the languages in the chunks or creating duplicate chunks with each language. Similarly for data PIDs, these are either muxed together or multiple versions of chunks are needed with different data PIDs.
- There is no standard aggregate format for HLS, which means that many chunk files are created. A day's worth of one channel with eight profiles at 10-second chunk duration will consist of almost 70,000 files. Managing such a large collection of files is not convenient.

Note: iOS 5 offers features that loosen some of these limitations, but the distribution of iOS versions means that these disadvantages persist in the market today.

Microsoft Silverlight Smooth Streaming (MSS)

Silverlight Smooth Streaming delivers streams as a sequence of ISO MPEG-4 files (see [MSS] and [MP4]). These are typically pushed by an encoder to a Microsoft IIS server (using HTTP POST), which aggregates them for each profile into an 'ismv' file for video and an 'isma' file for audio. The IIS server also creates an XML manifest file that contains information about the bitrates and resolutions of the available profiles (see Figure 7). When the request for the manifest comes from a Microsoft IIS server, it has a specific format:

```
http://{serverName}/{PublishingPointPath}/{PublishingPointName}.isml/manifest
```

The `PublishingPointPath` and `PublishingPointName` are derived from the IIS configuration.

Unlike HLS, in which the URLs are given explicitly in the playlist, in MSS the manifest files contain information that allows the client to create a RESTful URL request based on timing

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1175
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

information in the stream. For live streaming, the client doesn't need to repeatedly download a manifest – it computes the URLs for the chunks in each profile directly. The segments are extracted from the ismv and isma files and served as 'fragmented' ISO MPEG-4 (fMP4) files. MSS (optionally) separates the audio and video into separate chunks and combines them in the player. The URLs below show typical requests for video and audio. The `QualityLevel` indicates the profile and the `video=` and `audio-eng=` indicate the specific chunk requested. The `Fragments` portion of the request is given using a time stamp (usually in hundred nanosecond intervals) that the IIS server uses to extract the correct chunk from the aggregate MP4 audio and/or video files.

```
http://sourcehost/local/2/mysticSmooth.isml/QualityLevels(350000)/Fragments(video=2489452460333)
http://sourcehost/local/2/mysticSmooth.isml/QualityLevels(31466)/Fragments(audio-eng=2489450962444)
```

In the VoD case, the manifest files contain timing and sequence information for all the chunks in the content. The player uses this information to create the URL requests for the audio and video chunks.

Note that the use of IIS as the source of the manifest and fMP4 files doesn't preclude use of standard HTTP servers in the CDN. The CDN can still cache and deliver the manifest and chunks as it would any other files. More information about MSS can be found at Microsoft (see [SSTO]) and various excellent blogs of the developers of the technology (see [SSBLOG]).

MSS Considerations

Some advantages of MSS are:

- IIS creates an aggregate format for the stream, so that a small number of files can hold all the information for the complete smooth stream.
- The use of IIS brings useful analysis and logging tools, as well as the ability to deliver more MSS and HLS content directly from the IIS server.
- The recommended use of a small chunk size allows for rapid adaptation during HTTP streaming playback.
- The segregated video and audio files mean that delivery of different audio tracks involves just a manifest file change.
- The aggregate file format supports multiple data tracks that can be used to store metadata about ad insertion, subtitling, etc.

Some disadvantages of MSS are:

- The need to place an IIS server in the data flow adds an extra point of failure and complicates the network.
- On PCs, MSS requires installation of a separate Silverlight plug-in.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1176
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
<SmoothStreamingMedia MajorVersion="2" MinorVersion="0" TimeScale="10000000" Duration="0" LookAheadFragmentCount="2"
IsLive="TRUE" DVRWindowLength="300000000">
    <StreamIndex Type="video" QualityLevels="7" TimeScale="10000000" Name="video" Chunks="14"
        Url="QualityLevels({bitrate})/Fragments(video={start time})" MaxWidth="1280" MaxHeight="720"
        DisplayWidth="1280" DisplayHeight="720">
        <QualityLevel Index="0" Bitrate="350000"
        CodecPrivateData="00000001274D401F9A6282833F3E022000007D20001D4C12800000000128EE3880" MaxWidth="320"
        MaxHeight="180" FourCC="H264" NALUnitLengthField="4"/>
        <QualityLevel Index="1" Bitrate="500000"
        CodecPrivateData="00000001274D401F9A628343F6022000007D20001D4C12800000000128EE3880" MaxWidth="416"
        MaxHeight="240" FourCC="H264" NALUnitLengthField="4"/>
        <QualityLevel Index="2" Bitrate="750000"
        CodecPrivateData="00000001274D401F9A6281405FF2E022000007D20001D4C12800000000128EE3880" MaxWidth="640"
        MaxHeight="360" FourCC="H264" NALUnitLengthField="4"/>
        <QualityLevel Index="3" Bitrate="1000000"
        CodecPrivateData="00000001274D401F9A6281405FF2E022000007D20001D4C12800000000128EE3880" MaxWidth="640"
        MaxHeight="360" FourCC="H264" NALUnitLengthField="4"/>
        <QualityLevel Index="4" Bitrate="1250000"
        CodecPrivateData="00000001274D40289A6281B07FF36022000007D20001D4C12800000000128EE3880" MaxWidth="864"
        MaxHeight="486" FourCC="H264" NALUnitLengthField="4"/>
        <QualityLevel Index="5" Bitrate="1500000"
        CodecPrivateData="00000001274D40289A6281E022FDE022000007D20001D4C12800000000128EE3880" MaxWidth="960"
        MaxHeight="540" FourCC="H264" NALUnitLengthField="4"/>
        <QualityLevel Index="6" Bitrate="3000000"
        CodecPrivateData="00000001274D40289A6280A00B760220000007D20001D4C12800000000128EE3880" MaxWidth="1280"
        MaxHeight="720" FourCC="H264" NALUnitLengthField="4"/>
        <c t="2489302977667"/>
        <c t="2489324332333"/>
        <c t="2489345687000"/>
        <c t="2489367041667"/>
        <c t="2489388396333"/>
        <c t="2489409751000"/>
        <c t="2489431105667"/>
        <c t="2489452460333"/>
        <c t="2489473815000"/>
        <c t="2489495169667"/>
        <c t="2489516524333"/>
        <c t="2489537879000"/>
        <c t="2489559233667"/>
        <c t="2489580588333" d="21354667"/>
    </StreamIndex>
    <StreamIndex Type="audio" QualityLevels="4" TimeScale="10000000" Language="eng" Name="audio_eng" Chunks="14"
        Url="QualityLevels({bitrate})/Fragments(audio_eng={start time})">
        <QualityLevel Index="0" Bitrate="31466" CodecPrivateData="1190" SamplingRate="48000" Channels="2"
        BitsPerSample="16" PacketSize="4" AudioTag="255" FourCC="AACL"/>
        <QualityLevel Index="1" Bitrate="31469" CodecPrivateData="1190" SamplingRate="48000" Channels="2"
        BitsPerSample="16" PacketSize="4" AudioTag="255" FourCC="AACL"/>
        <QualityLevel Index="2" Bitrate="31472" CodecPrivateData="1190" SamplingRate="48000" Channels="2"
        BitsPerSample="16" PacketSize="4" AudioTag="255" FourCC="AACL"/>
        <QualityLevel Index="3" Bitrate="31481" CodecPrivateData="1190" SamplingRate="48000" Channels="2"
        BitsPerSample="16" PacketSize="4" AudioTag="255" FourCC="AACL"/>
        <c t="2489301415778"/>
        <c t="2489322749111"/>
        <c t="2489344082444"/>
        <c t="2489365415778"/>
        <c t="2489386749111"/>
        <c t="2489408295778"/>
        <c t="2489429629111"/>
        <c t="2489450962444"/>
        <c t="2489472295778"/>
        <c t="2489493629111"/>
        <c t="2489514962444"/>
        <c t="2489536295778"/>
        <c t="2489557629111"/>
        <c t="2489578962444" d="21333334"/>
        </StreamIndex>
</SmoothStreamingMedia>
```

Figure 7. A sample MSS manifest file. The elements with "t="248..."' specify the time stamps of chunks that the server already has and can deliver. These are converted to Fragment timestamps in the URL requesting an MP4 chunk. The returned chunk holds time stamps of the next chunk or two (in its UUID box), so that the client can continue to fetch chunks without having to request a new manifest.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1177
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Adobe HTTP Dynamic Streaming (HDS)

Adobe HDS was defined after both HLS and MSS and makes use of elements in each (see [HDS]). In HDS, an XML manifest file (of file type f4m) contains information about the available profiles (see Figure 8 and [F4M]). As in HLS, data that allows the client to derive the URLs of the available chunks is repeatedly downloaded by the client; in HDS this is called the bootstrap information. The bootstrap information is in a binary format and hence isn't human-readable. As in MSS, segments are encoded as fragmented MP4 files that contain both audio and video information in one file. HDS chunk requests have the form:

```
http://server_and_path/QualityModifierSeg'segment_number'-Frag'fragment_number'
```

where the segment and fragment number together define a specific chunk. As in MSS, an aggregate (f4f) file format is used to store all the chunks and extract them when a specific request is made.

```
<manifest>
    <id>USP</id>
    <startTime>2006-07-24T07:15:00+01:00</startTime>
    <duration>0</duration>
    <mimeType>video/mp4</mimeType>
    <streamType>live</streamType>
    <deliveryType>streaming</deliveryType>
    <bootstrapInfo profile="named" url="mysticHDS.bootstrap"/>
    <media url="mysticHDS-audio_eng=31492-video=3000000-" bitrate="3031" width="1280" height="720"/>
    <media url="mysticHDS-audio_eng=31492-video=1500000-" bitrate="1531" width="960" height="540"/>
    <media url="mysticHDS-audio_eng=31492-video=1250000-" bitrate="1281" width="864" height="486"/>
    <media url="mysticHDS-audio_eng=31492-video=1000000-" bitrate="1031" width="640" height="360"/>
    <media url="mysticHDS-audio_eng=31492-video=750000-" bitrate="781" width="640" height="360"/>
    <media url="mysticHDS-audio_eng=31492-video=500000-" bitrate="531" width="416" height="240"/>
    <media url="mysticHDS-audio_eng=31469-video=350000-" bitrate="381" width="320" height="180"/>
</manifest>
```

Figure 8. A sample HDS manifest file.

HDS Considerations

Some advantages of HDS are:

- The Flash client is available on multiple devices and is installed on almost every PC in the world.
- HDS is a part of Flash and can make use of Flash's environment and readily available developer base.

Some disadvantages of HSS are:

- HDS is a relative late-comer and could suffer more from stability issues.
- Adobe's Flash access roadmap is rapidly changing, making it difficult to deploy a stable ecosystem.
- Adobe holds close the details of its format. Combined with the binary format of the bootstrap file, this limits the ecosystem of partners offering compatible solutions.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1178
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4. Feature Comparison

In this section, we compare HLS, HDS and MSS usability for a variety of common usage scenarios.

Delivery of Multiple Audio Channels

In HLS, TS files can easily carry multiple audio tracks, but this isn't necessarily a strength when it comes to adaptive HTTP streaming because it means the TS chunks are bigger, even if only one audio stream is consumed. In places where there are multiple languages, multiple audio streams included in each chunk can consume a larger portion of the bandwidth than the video. For HLS to work well, packagers have to create chunks with each video/audio language combination, increasing the storage needed for holding the chunks.

MSS stores each audio track as a separate file, so it's easy to create chunks with any video-audio combination. The total number of files is the same as with HLS, but the ability to store the audio and video once in aggregate format makes MSS more efficient.

HDS does not offer a compelling solution to delivering different audio streams.

Encryption and DRM

HLS supports encryption of each TS file. This means that all the data in the TS file is encrypted and there is no way to extract it without access to the decryption keys. All metadata related to the stream (e.g. the location of the decryption keys) must be included in the playlist file. This works well, except that HLS does not specify a mechanism for authenticating clients to receive the decryption keys. This is considered a deployment issue. A number of vendors offer HLS-type encryption, often with their own twist which makes the final deployment incompatible with other implementations.

MSS uses Microsoft's PlayReady, which gives a complete framework for encrypting content, managing keys, and delivering them to clients. In PlayReady, only the payload of the fMP4 file is encrypted, so the chunk can carry other metadata. Microsoft makes PlayReady code available to multiple vendors that productize it, and so a number of vendors offer PlayReady capability (in a relatively undifferentiated way).

HDS uses Adobe's Flash Access, which has an interesting twist that simplifies interaction between the key management server and the scrambler that does the encryption. Typically, keys must be exchanged between these two components, and this exchange interface is not standardized. Each DRM vendor/scrambler vendor pair must implement this pair-wise proprietary API. With Adobe Access, no key exchange is necessary – the decryption keys are sent along with the content, but are themselves encrypted. Access to those keys is granted at run time, but no interaction between the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1179
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

key management system and scrambler is needed. Adobe licenses its Access code to third parties, or it may be used as part of the Flash Media Server product suite.

Closed Captions / Subtitling

HLS can decode and display closed captions (using ATSC Digital Television Standard Part 4 – MPEG-2 Video System Characteristics - A/53, Part 4:2007, see [ATCA]) included in the TS chunks as of iOS 4.3 (the implementation in iOS 4.2 is more problematic). For DVB teletext, packagers need to convert the subtitle data into ATSC format or wait for clients to support teletext data.

MSS supports data tracks that hold Time Text Markup Language (TTML), a way to specify a separate data stream with subtitle, timing and placement information (see [TTML]). For MSS, packagers need to extract subtitle information from their input and convert it into a TTML track. Microsoft's implementation of MSS client currently offers support for W3C TTML, but not for SMPTE TTML (see [SMPTE-TT]), which adds support for bitmapped subtitles, commonly used in Europe.

HDS supports data tracks that hold subtitles as DFXP file data, based on the TTML format. Clients can selectively download this data, similar to MSS, but client support requires customization and additional development.

Targeted Ad insertion

HLS is the simplest protocol to use for chunk-substitution-based ad insertion. With HLS, the playlist file can be modified to deliver different ad chunks to different clients (see Figure 9). The `EXT-X-DISCONTINUITY` tag can be used to tell the decoder to reset (e.g. because subsequent chunks may have different PID values), and only the sequence ID must be managed carefully, so that the IDs line up when returning to the main program. HDS also supports a repeated download of bootstrap data used to specify chunks, and this can be modified to create references to ad chunks – but because the bootstrap data format is binary, and the URLs are RESTful with references to chunk indexes, the process is complex.

MSS is trickier when it comes to chunk-based ad insertion for live streams. The fact that chunks contain timing information used to request the next chunk means that all ad chunks have to have identical timing to the main content chunks (or that some other method is used to reconcile the program time when returning to the program). Nevertheless, a proxy can be used to redirect RESTful URL chunk requests and serve different chunks to different clients.

Both MSS and HDS can deliver control events in separate data tracks. These can be used to trigger client behaviors using the Silverlight and Flash environments, including ad insertion behavior. This is beyond the scope of this paper, which is focused on 'in stream' insertion. One difference between MSS and HDS is that MSS defines a client-side ad insertion architecture, whereas HDS does not.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1180
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

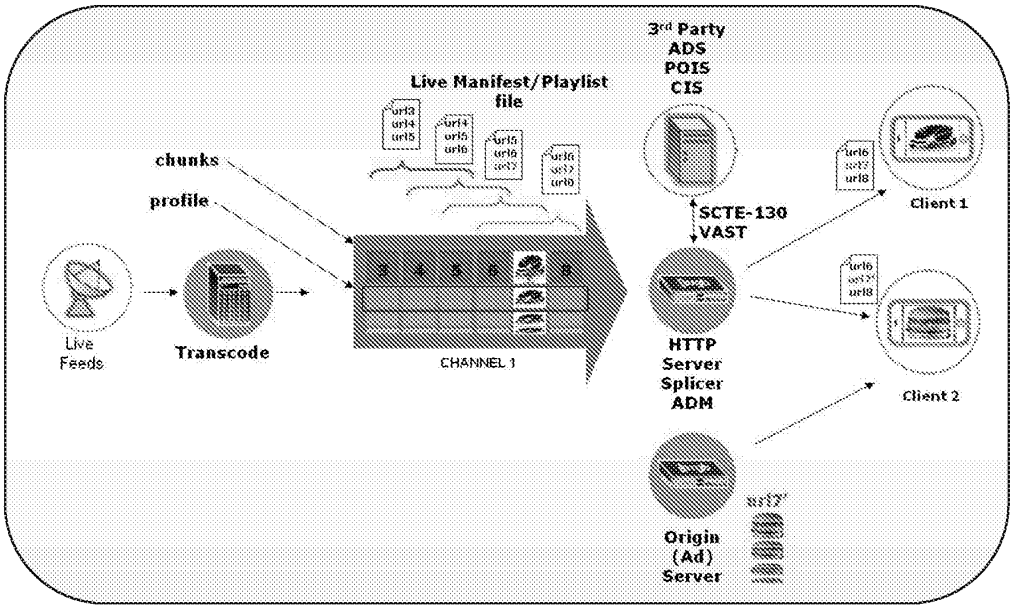


Figure 9. HLS ad insertion in which changes to the playlist file delivered to each client cause each client to make different URL requests for ads and thus receive targeted ad content.

Trick Modes (Fast-forward / Rewind)

VoD trick modes, such as fast-forward or rewind, are messy in all the protocols. None of the protocols offer native support for fast-forward or rewind. Some MSS variations have defined zoetrope images that can be embedded in a separate track. These can be used to show still images from the video sequence and allow seeking into a specific location in the video.

HDS supports a fast playback mode, but this doesn't appear to work well.

Custom VoD Playlists

It is convenient to be able to take content from multiple different assets and stitch them together to form one asset. This is readily done in HLS, where the playlist can contain URLs that reference chunks from different encodings and locations. In MSS and HDS, the RESTful URL name spaces and the references to chunks via time stamp or sequence number makes such playlists basically impossible to construct.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1181
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Fast Channel Change

Adaptive HTTP streaming can download low bitrate chunks initially, making channel 'tune in' times low. The duration of the chunk directly affects how fast the channel adapts to a higher bandwidth (and higher quality video). Because of this, MSS and HDS, which are tuned to work with smaller chunks, tend to work a bit better than HLS.

Failover Due to Upstream Issues

HLS manifests can list failover URLs in case content is not available. The mechanism used in the variant playlist file to specify different profiles can be used to specify failover servers, since the client (starting with iOS 3.1 and later) will attempt to change to the next profile when a profile chunk requests returns an HTTP 404 'file not found' code. This is a convenient, distributed redundancy mode.

MSS utilizes a run-time client that is fully programmable. Similarly, HDS works within the Flash run-time environment. That means that the some failover capabilities can be built into the client. However, in both cases, there isn't a built-in mechanism in the protocol to support a generic failover capability.

All the protocols will failover to a different profile if chunks/playlists in a given profile are not available. This potentially allows any of the protocols to be used in a "striping" scenario in which alternate profiles come from different encoders (as long as the encoders output IDR aligned streams), so that an encoder failure causes the client to adapt to a different, available profile.

Stream Latency

Adaptive HTTP clients buffer a number of segments. Typically one segment is currently playing, one is cached, and a third is being downloaded – so that the end-to-end latency is minimally about three segment durations long. With HLS recommended to run with 10-second chunks (though this isn't necessary), this latency can be quite long.

Of the three protocols, only MSS has implemented a low latency mode in which sub-chunks are delivered to the client as soon as they are available. The client doesn't need to wait for a whole chunk's worth of stream to be available at the packager before requesting it, reducing its overall end-to-end latency.

Ability to Send Other Data to the Client (Including Manifest Compression)

HLS and HDS can send metadata to the client in their playlist and manifest files. MSS and HDS allow data tracks which can trigger client events and contain almost any kind of data. HLS allows a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1182
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

separate ID3 data track to be muxed into the TS chunks. This can be used to trigger client-side events.

MSS also allows manifest files to be compressed using gzip (as well as internal run-length-type compression constructs) for faster delivery.

5. Conclusion

A review of how these three formats compare is shown in the table below. In spite of MSS's better performance, the technology that will gain the most market share remains to be seen. Ultimately, it may be DASH that succeeds in the market in the long run and not any of the technologies discussed here.

Feature	HLS	MSS	HDS
Multiple audio channels		☺	
Encryption		☺	☺
Closed captions / subtitling	☺	☺	
Custom VoD playlists	☺		
ability to insert ads	☺	😐	
trick modes (fast forward / rewind)		😐	
fast channel change & Stream latency		☺	☺
Client failover	☺		
Metadata	☺	☺	☺

In any case, online and mobile viewing of premium video content is rapidly complementing the traditional TV experience, and delivery over the Internet requires new protocols to produce a high quality of experience based on device type and network congestion. Apple HLS, Microsoft Smooth Streaming and Adobe Flash HDS represent adaptive delivery protocols that enable high-quality video consumption experiences over the Internet. Content providers must now equip network delivery infrastructure with products capable of receiving standard video containers, slicing them into segments, and delivering those segments along with encryption where required. As with all new technology, the choice of delivery protocol will be made based on a combination of technical and business factors.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1183
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6. References

[HLS] HTTP Live Streaming, R. Pantos, http://tools.ietf.org/html/draft-pantos-http-live-streaming-06

[HLS1] HTTP Live Streaming, http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/HTTPLiveStreaming/_index.html

[MP4] International Organization for Standardization (2003). "MPEG-4 Part 14: MP4 file format; ISO/IEC 14496-14:2003"

[SSBLOG] http://blog.johndeutscher.com/, http://blogs.iis.net/samzhang/ , http://alexzambelli.com/blog/, http://blogs.iis.net/jboch/

[SSTO] Smooth Streaming Technical Overview, http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/

[ATCA] ATSC Digital Television Standard Part 4 – MPEG-2 Video System Characteristics (A/53, Part 4:2007), http://www.atsc.org/cms/standards/a53/a_53-Part-4-2009.pdf

[TTML] Timed Text Markup Language, W3C Recommendation 18 November 2010, http://www.w3.org/TR/ttaf1-dfxp/

[SMPTE-TT] SMPTE Time Text format, https://www.smpte.org/sites/default/files/st2052-1-2010.pdf

[MSS] IIS Smooth Streaming Transport Protocol, http://www.iis.net/community/files/media/smoothspecs/%5BMS-SMTH%5D.pdf

[F4M] Flash Media Manifest File Format Specification, http://osmf.org/dev/osmf/specpdfs/FlashMediaManifestFileFormatSpecification.pdf

[HDS] HTTP Dynamic Streaming on the Adobe Flash Platform, http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf

[AHS] 3GPP TS 26.234: "Transparent end-to-end packet switched streaming service (PSS); Protocols and codecs".

[HAS] OIPF Release 2 Specification - HTTP Adaptive Streaming http://www.openiptvforum.org/docs/Release2/OIPF-T1-R2-Specification-Volume-2a-HTTP-Adaptive-Streaming-V2_0-2010-09-07.pdf

[HLSID3] Timed Metadata for HTTP Live Streaming, http://developer.apple.com/library/ios/#documentation/AudioVideo/Conceptual/HTTP_Live_Streaming_Metadata_Spec/Introduction/Introduction.html

[DVB-BITMAPS] Digital Video Broadcasting (DVB); Subtitling systems, ETSI EN 300 743 V1.3.1 (2006-11), http://www.etsi.org/deliver/etsi_en/300700_300799/300743/01.03.01_60/en_300743v010301p.pdf

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1184
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

streamingmedia.com
streaming media
Download TODAY!
HOME EUROPEVENTS PRODUCER SUBSCRIBE ARTICLES VIDEO TOPICS VERTICALS RESOURCES WEB EVENTS CONFERENCES ABOUT
streaming media
Download it Today!
SEARCH STREAMINGMEDIA.COM

HTTP Streaming: What You Need to Know

Understanding this hybrid delivery approach is key to advancing robust streaming solutions.

By *Tim Siglin*

For the rest of the February/March 2010 (Sourcebook) issue of Streaming Media magazine please *click here*

SOCIALBOOKMARK
Email
Print
Digg
Send
Like 1
Tweet 1
Share 0
+1 0

Companies Mentioned:

Adobe
Akamai
Microsoft
octoshape

[This article originally appeared in the 2010 Streaming Media Sourcebook.*]*

In the world of streaming, there are many terms that create confusion, starting with the word "streaming," which is often used interchangeably with the term "progressive download." Even the words codec, format, system, and wrapper/container are often used interchangeably, albeit incorrectly.

Of all of the terms, though, the one that seems to have the most uncertainty around it is HTTP streaming. As recently as 2007, the term was easily dismissed as a mistake because HTTP was a protocol that allowed content to be routed through port 80 while streaming often required a separate port.

In the past 2 years, however, HTTP streaming has become a reality, both for live and on-demand content. It is now, in turn, often confused with another technology that can accompany HTTP streaming—adaptive bitrate streaming.

In this article, I'll address the differences between HTTP streaming, generic streaming, and progressive downloads. I'll also touch a bit on adaptive bitrate technologies, even though these technologies are a bit nascent and several of the major players have yet to fully firm up the technologies' integration into HTTP streaming.

Defining HTTP Streaming

As such, a set of definitions is in order from the outset. First, let's look at progressive downloads, streaming media, and hybrid solutions.

Progressive Download

A progressive download is a pre-encoded file that has been set to begin playing prior to the completion of the full download. In the old days, a file would completely download before it began playing, which meant the user had to wait several minutes (or hours) before he or she could watch the video.

The reason for the confusion of progressive downloading with streaming is that progressive downloaded file playback behaves somewhat like a streaming file, in that it often begins playing within 3–10 seconds of the time the file is selected. Streaming often requires a buffer, as does progressive downloading, to address network inconsistencies; playback begins only after an initial playback buffer is loaded.

With streaming, a user can click on any location along a file's timeline and begin to watch the video after a few seconds of buffering. Progressive downloads, on the other hand, require the file to be downloaded linearly, meaning that a user's bandwidth may constrain the user from jumping ahead several minutes, at least until that portion of the video file has been downloaded.

To address this limitation in progressive downloads and to allow commands such as fast-forward, rewind, pause, or location seek, some progressive download solutions offer "trick mode" requests that will divert downloading to the portion of the file that the user has requested. To do this, the file needs to be segmented, which we'll later see is critical to the development of adaptive bitrate streaming and HTTP streaming delivery.

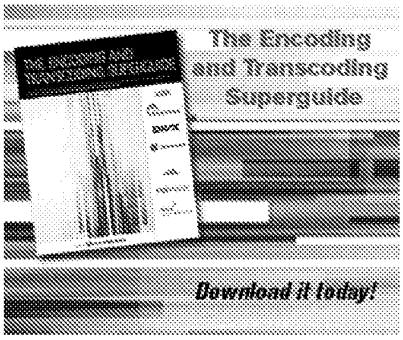
Streaming Media

Unlike progressive downloads, which can often be downloaded from an HTTP server just as a PDF document can be delivered via HTTP, streaming content has, until now, required the use of a separate protocol to deliver streaming packets.

The long-standard real-time protocol (RTP), which was used by videoconferencing and voice-over-IP solutions, was joined by a companion, streaming-specific protocol (RTSP) as a way to differentiate traffic



UPCOMING INDUSTRY CONFERENCES
Streaming Forum 2014 (London) [24-25 June 2014]
Streaming Media West 2014 [18-19 November 2014]
Streaming Media Producer Live [18-19 November 2014]
Recent Conferences
Streaming Media East 2014 [13-14 May 2014]
Content Delivery Summit 2014 [12 May 2014]
Streaming Media West 2013 [19-20 November 2013]
Streaming Media Producer Live 2013 [19-20 November 2013]
Past Conferences



POPULAR ARTICLES NEWS FORUMS
TOP 10 MOST POPULAR ARTICLES
How to Create Interactive HTML5 Video
eBay Embraces Enterprise YouTube
WD Intros WD TV, Its First Set-Top Box with Miracast Support
The ROI of Enterprise Webcasting: Justifying the Expense
MPEG LA's HEVC Licensing Terms Are Flawed, Will Prevent Adoption
Anatomy of a Viral Video: The First Days Are the Most Important
How Be Inspired Films Shoots

from HTTP content, which most often used port 80 and was not as time-sensitive as streaming content. Unfortunately, the use of a port other than port 80 for streaming also meant it was easily identified, which in turn meant that streaming content could be blocked at the router by IT departments intent on keeping streaming content from "damaging" their network.

Besides the trick modes previously mentioned, there is another innovation in streaming that allows content to be rapidly launched. Known as "fast start" or "hinting," this rapid streaming innovation allowed the streaming server to allow the player to begin playback without a buffer, almost guaranteeing that streaming content would begin playing back faster than progressive download content.

Page 1 of 5 next »

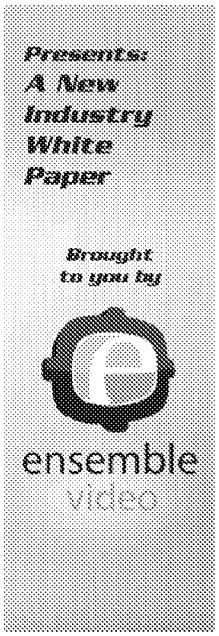
Add a comment...

Comment using...

Facebook social plugin

Related Articles

Online Video News: Encoding.com Now Supports Apple HTTP Streaming and the iPad
Cloud-based encoding service Encoding.com has added HTTP Live Streaming, with presets for iPhone, iPod Touch, and now the iPad.





***Streaming Media* Magazine**

June 2014
Subscribe

Current Issue **Past Issues**

- How to Create Interactive HTML5 Video
- How to Develop a Viable 4K Production Workflow
- How Be Inspired Films Shoots Conferences the TED Way
- Twitch Is a Billion Dollar Surprise, Even for Video Insiders
- Don't Just Compress the Video, Compress the Content

Current Digital Edition



SPONSORED LINKS

Looking for Trusted Suppliers? Check out the Industry Directory here

INSIDE STREAMINGMEDIA.COM

Blog
SM Magazine
SM Digital Edition
Streaming Media App
Streaming Media TV
Conference Videos
Video Tutorials
Readers Choice
Research Reports
Press Releases
Web Events
About Us
Newsletters
Industry News
SM White Papers
Industry White Papers
Sponsored Articles
Forums
Discussion Lists
Industry Directory
Industry Jobs
Contact Us
What is Streaming

RESEARCH CENTERS

TOPICS
Mobile Video
Music/Radio
Online Video Platforms
OTT Video
Video Advertising
Video Content Protection/DRM
Video Creation/Production
Video Encoding/Transcoding
Video Formats/Players/Codecs
Video Hosting/Delivery
Video Legal Issues
Video SEO

VERTICALS
Education Video
Enterprise Video
Government Video
Media/Entertainment Video
Online Video Industry

OTHER IT/ SITES

Streaming Media Europe
Streaming Media Producer
destinationCRM
ITContent
Faulkner Information Services
Information Today
Information Today Europe
ITResearch.com
KMWorld
OnlineVideo.net
Smart Customer Service
Speech Technology

© 2000 - 2014, StreamingMedia.com, an Information Today, Inc. (ITI) company

[PRIVACY/COOKIES POLICY]

streaming media research Sponsored by piksel

HTTP Streaming: What You Need to Know

Understanding this hybrid delivery approach is key to advancing robust streaming solutions.

By *Tim Siglin*

For the rest of the February/March 2010 (Sourcebook) issue of Streaming Media magazine please *click here*

« previous **Page 2 of 5** next »

SOCIALANNOTATE
Email
Print
Digg
Send
Like 1
Tweet 1
Share 0
+1 0

Companies Mentioned:

Adobe
Akamai
Microsoft
octoshape

One example of this is Instant-On, Apple's patent-pending streaming technology, which offers content that begins to play as soon its link is clicked. This solution has direct bearing on Apple's HTTP streaming solutions of adaptive bitrate content.

Another piece to this puzzle that has been used in the past to address network inconsistencies is the encoding of multiple bitrates for each video file. The initial playback request included a bandwidth measurement, which triggered the selection of a discrete file that most closely matched the user's anticipated bitrate (rounded down to the next lowest bitrate).

This solution was effective in some instances, although it assumes that the bandwidth of the initial request is a consistent bandwidth.

Stuttering is the stalling of content delivery that occurs if the streaming bandwidth falls below the stream's optimal bitrate. Even multiple bitrate streaming was susceptible to this because the initial bandwidth may not equal the sustained bandwidth, meaning that a sustainable viewing experience cannot be maintained.

For on-demand content, stuttering is annoying. Content delivery is momentarily stopped and is later picked up at that point in the stream once the buffer is replenished. For live content, however, this stuttering is deadly, as the video or audio stream continues while the viewer is waiting for his or her network to recover enough to resume the broadcast. Once the network recovers, the broadcast stream skips ahead to pick up the live broadcast in real time.

Yet, as I'll cover later in this article, the basis of multibitrate streaming is also key to emerging HTTP streaming solutions.

Finally, streaming has some efficiencies in terms of delivery, including the ability to eliminate excessive downloading of long-form content that a user may abandon partway through. When it comes to progressive downloads, users with very fast connections may be able to download an entire video file in less than half the time it takes to view the file. Should a user then choose to abandon the content halfway through the viewing time, the content delivery has "wasted" almost 50% of the bits by downloading much more content than the user watched.

Streaming, on the other hand, only delivers content until the user abandons viewing the content, limiting the number of wasted bits. In recent years, progressive downloads have gained the ability to be throttled, meaning that only a slightly higher bandwidth download is allowed regardless of the user's possible bandwidth, which also limits the number of wasted bits should a user abandon viewing before the end of a file. The downside of progressive download throttling is that it often limits the ability for viewers to skip ahead by minutes rather than by a few seconds.

This efficiency is also now wrapped into HTTP streaming if adaptive bitrates are used.

One other efficiency in streaming is that files are not saved on the viewer's hard drive. This is often viewed as primarily beneficial from a rights management or copyright standpoint. But the delivery efficiency is that video length has no bearing on the amount of user hard disk space required to view a video file.

As streaming content has to be restreamed to be viewed a second time, the throttling of progressive download content often also means that this content will be rapidly jettisoned, requiring redownloading for a second viewing of the same content, unless the second viewing is done immediately following the first viewing.

SEARCH STREAMINGMEDIA.COM



UPCOMING INDUSTRY CONFERENCES
Streaming Forum 2014 (London) [24-25 June 2014]
Streaming Media West 2014 [18-19 November 2014]
Streaming Media Producer Live [18-19 November 2014]
Recent Conferences
Streaming Media East 2014 [13-14 May 2014]
Content Delivery Summit 2014 [12 May 2014]
Streaming Media West 2013 [19-20 November 2013]
Streaming Media Producer Live 2013 [19-20 November 2013]
Past Conferences



POPULAR ARTICLES NEWS FORUMS
TOP 10 MOST POPULAR ARTICLES
How to Create Interactive HTML5 Video
eBay Embraces Enterprise YouTube
WD Intros WD TV, Its First Set-Top Box with Miracast Support
The ROI of Enterprise Webcasting: Justifying the Expense
MPEG LA's HEVC Licensing Terms Are Flawed, Will Prevent Adoption
Anatomy of a Viral Video: The First Days Are the Most Important
How Be Inspired Films Shoots

Post a comment...

Comment using...

Facebook social plugin

Related Articles

Online Video News: Encoding.com Now Supports Apple HTTP Streaming and the iPad
Cloud-based encoding service Encoding.com has added HTTP Live Streaming, with presets for iPhone, iPod Touch, and now the iPad.





***Streaming Media* Magazine**

June 2014
Subscribe

Current Issue **Past Issues**

- How to Create Interactive HTML5 Video
- How to Develop a Viable 4K Production Workflow
- How Be Inspired Films Shoots Conferences the TED Way
- Twitch Is a Billion Dollar Surprise, Even for Video Insiders
- Don't Just Compress the Video, Compress the Content

Current Digital Edition



SPONSORED LINKS

Looking for Trusted Suppliers? Check out the Industry Directory here

INSIDE STREAMINGMEDIA.COM

Blog
SM Magazine
SM Digital Edition
Streaming Media App
Streaming Media TV
Conference Videos
Video Tutorials
Readers Choice
Research Reports
Press Releases
Web Events
About Us
Newsletters
Industry News
SM White Papers
Industry White Papers
Sponsored Articles
Forums
Discussion Lists
Industry Directory
Industry Jobs
Contact Us
What is Streaming

RESEARCH CENTERS

TOPICS
Mobile Video
Music/Radio
Online Video Platforms
OTT Video
Video Advertising
Video Content Protection/DRM
Video Creation/Production
Video Encoding/Transcoding
Video Formats/Players/Codecs
Video Hosting/Delivery
Video Legal Issues
Video SEO

VERTICALS
Education Video
Enterprise Video
Government Video
Media/Entertainment Video
Online Video Industry

OTHER IT/ SITES

Streaming Media Europe
Streaming Media Producer
destinationCRM
ITContent
Faulkner Information Services
Information Today
Information Today Europe
ITResearch.com
KMWorld
OnlineVideo.net
Smart Customer Service
Speech Technology

© 2000 - 2014, StreamingMedia.com, an Information Today, Inc. (ITI) company

[PRIVACY/COOKIES POLICY]

streamingmedia.com
streaming media Download Today!
HOME EUROPEAN PRODUCER CLASSROOM ARTICLES VIDEO TOPICS VERTICALS RESOURCES WEB EVENTS CONFERENCES ABOUT
streaming media See it Now and Subscribe FREE*!
*Free access to qualified US subscribers. Digital free to all, print for subscribers.

HTTP Streaming: What You Need to Know

Understanding this hybrid delivery approach is key to advancing robust streaming solutions.

By *Tim Siglin*

For the rest of the February/March 2010 (Sourcebook) issue of Streaming Media magazine please *click here*

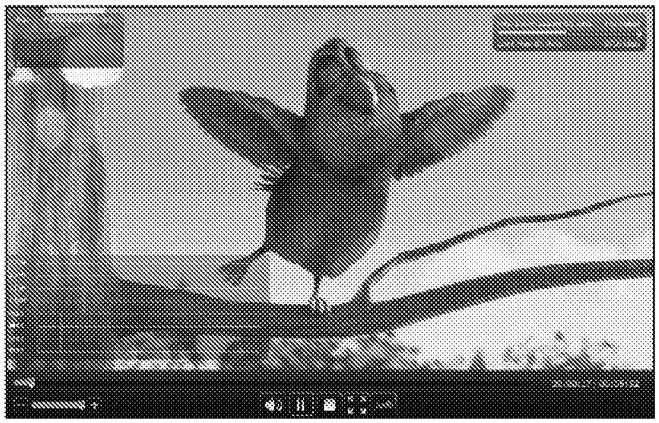
SOCIALBOOKMARK
Email
Print
Digg
Send
Like 1
Tweet 1
Share 0
+1 0

Companies Mentioned:

Adobe
Akamai
Microsoft
octoshape

Hybrid solutions

Besides the progress of streaming and progressive download features over the past few years, which brings basic functionality between the two types of delivery within a very narrow range, additional hybrid solutions have been proposed.



Microsoft Smooth Streaming *Microsoft s Smooth Streaming combines HTTP delivery and adaptive bitrate technology.*

Three major needs have pushed these hybrid solutions forward: efficiency, scalability, and ubiquity.

On the efficiency front, there are numerous past solutions that wrap RTSP streaming video traffic into HTTP packets

SEARCH STREAMINGMEDIA.COM
streaming media See it Now and Subscribe FREE*!
UPCOMING INDUSTRY CONFERENCES
Streaming Forum 2014 (London) [24-25 June 2014]
Streaming Media West 2014 [18-19 November 2014]
Streaming Media Producer Live [18-19 November 2014]
Recent Conferences
Streaming Media East 2014 [13-14 May 2014]
Content Delivery Summit 2014 [12 May 2014]
Streaming Media West 2013 [19-20 November 2013]
Streaming Media Producer Live 2013 [19-20 November 2013]
Past Conferences
Wowza Media Pioneer
Streaming. Simplified.
Wowza on Amazon Marketplace
» Learn more
POPULAR ARTICLES NEWS FORUMS
TOP 10 MOST POPULAR ARTICLES
How to Create Interactive HTML5 Video
eBay Embraces Enterprise YouTube
WD Intros WD TV, Its First Set-Top Box with Miracast Support
The ROI of Enterprise Webcasting: Justifying the Expense
MPEG LA's HEVC Licensing Terms Are Flawed, Will Prevent Adoption
Anatomy of a Viral Video: The First Days Are the Most Important
How Be Inspired Films Shoots

HTTP packets.

"RTSP streaming [via] HTTP packets … appear identical to standard web traffic and therefore are permitted through most firewalls," Apple stated in a document about the difference between RTSP and HTTP streaming. "This involves [an] extra layer of overhead however, resulting in a greater demand for bandwidth."

Flash HTTP streaming is another example. Up until Adobe's announcement of its HTTP/adaptive bitrate solution (which had not launched as of the time of this writing) anyone referring to Flash as HTTP streaming actually meant HTTP tunneling.

The reasoning for this misnomer is fairly simple: The Flash server that used the real-time media protocol (RTMP) was designed to be delivered on TCP rather than the UDP delivery of the more traditional RTSP steaming protocol. Given RTMP's ability to deliver via TCP, Adobe integrated RTMP tunneling via HTTP for both live and on-demand content. This appears equivalent to HTTP streaming. But the tunneling, in fact, causes a bit of an extra delay that might not be acceptable in all situations.

Based on the efficiency limitations, both Adobe and Apple have chosen to move toward true HTTP streaming solutions. Apple debuted HTTP streaming, sans adaptive bitrate streaming, for the iPhone when the phone launched in 2007.

Some content delivery network (CDN) vendors refer to their HTTP video file downloads or HTTP progressive downloads by the term HTTP streaming.

In terms of scalability, the need to consistently scale content to television-level audiences on any given day, without requiring weeks of preplanning for every web television event, has led to discussions of solutions that will allow streaming to scale to these levels. At the end of the article, I'll discuss multicasting and peer-to-peer delivery as long-term solutions. The short-term solution is a combination of the benefits of streaming and progressive downloading.

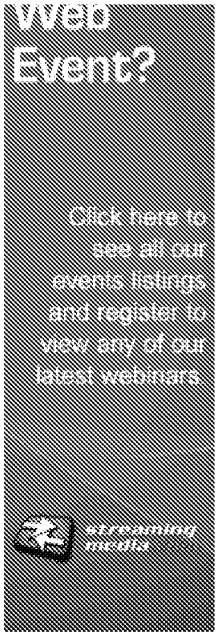
Finally, on the ubiquity front, as previously mentioned, much RTP/RTSP traffic is blocked by a high number of firewalls, making it difficult to deliver streams reliably. Even instant messaging audio and video conferencing capabilities, such as those used by Skype or iChat, rely on RTSP. Even beyond the typical firewall, however, is the ability to handle content delivery between different network types. Apple chose to go to HTTP streaming for the iPhone primarily based on the concept that RTSP video streaming on the iPhone would be difficult for a device that routinely moves between Global System for Mobile (GSM) and Wi-Fi networks.

« previous **Page 3 of 5** next »

Add a comment...
Comment using...
Facebook social plugin

Related Articles

Online Video News: Encoding.com Now Supports Apple HTTP Streaming and the iPad
Cloud-based encoding service Encoding.com has added HTTP Live Streaming, with presets for iPhone, iPod Touch, and now the iPad.





***Streaming Media* Magazine**

June 2014
Subscribe

Current Issue | Past Issues

- How to Create Interactive HTML5 Video
- How to Develop a Viable 4K Production Workflow
- How Be Inspired Films Shoots Conferences the TED Way
- Twitch Is a Billion Dollar Surprise, Even for Video Insiders
- Don't Just Compress the Video, Compress the Content

Current Digital Edition



SPONSORED LINKS

Looking for Trusted Suppliers? Check out the Industry Directory here

INSIDE STREAMINGMEDIA.COM

Blog
SM Magazine
SM Digital Edition
Streaming Media App
Streaming Media TV
Conference Videos
Video Tutorials
Readers Choice
Research Reports
Press Releases
Web Events
About Us
Newsletters
Industry News
SM White Papers
Industry White Papers
Sponsored Articles
Forums
Discussion Lists
Industry Directory
Industry Jobs
Contact Us
What is Streaming

RESEARCH CENTERS

TOPICS
Mobile Video
Music/Radio
Online Video Platforms
OTT Video
Video Advertising
Video Content Protection/DRM
Video Creation/Production
Video Encoding/Transcoding
Video Formats/Players/Codecs
Video Hosting/Delivery
Video Legal Issues
Video SEO

VERTICALS
Education Video
Enterprise Video
Government Video
Media/Entertainment Video
Online Video Industry

OTHER IT/ SITES

Streaming Media Europe
Streaming Media Producer
destinationCRM
ITContent
Faulkner Information Services
Information Today
Information Today Europe
ITIResearch.com
KMWorld
OnlineVideo.net
Smart Customer Service
Speech Technology

© 2000 - 2014, StreamingMedia.com, an Information Today, Inc. (ITI) company

[PRIVACY/COOKIES POLICY]

streamingmedia.com
streaming media
Download it Today!

streaming media
BEST GUIDELINES
Must Be Read on DVD
for your Digital Media
Encoding Farm

HTTP Streaming: What You Need to Know

Understanding this hybrid delivery approach is key to advancing robust streaming solutions.

By *Tim Siglin*

For the rest of the February/March 2010 (Sourcebook) issue of Streaming Media magazine please *click here*

« previous **Page 4 of 5** next »

SOCIALBOOKMARK
Email
Print
Digg
Send
Like 1
Tweet 1
Share 0
+1 0

Companies Mentioned:

Adobe
Akamai
Microsoft
octoshape

HTTP Streaming Meets Adaptive Streaming

Streaming via HTTP, without the additional benefits of progressive downloading, would be a bit of a nonstarter because the gestalt of benefits from both delivery options are important for efficiency, scalability, and ubiquity.

To realize the joint benefits, HTTP streaming solutions needed another aspect, which is called, generically, adaptive bitrate streaming. This hybrid media delivery method acts like streaming but uses progressive download—it uses port 80 like traditional HTTP traffic and delivers content in a series of small chunks.

This is not the multibitrate streaming of old as previously mentioned, where several full versions of a file were encoded and a single version was chosen at the outset of a user's viewing request. Each adaptive bitrate streaming file has a distinct bitrate, but each file is then segmented down to small "chunks" of 2–10 seconds, with all chunks aligned simultaneously between each of the files.

Like multibitrate streaming, adaptive bitrate streaming chooses a single file of a particular bitrate to initially stream. Once the bitrate is chosen, a single chunk (or several chunks, depending on the technology provider) begins to play immediately.

Before additional chunks are sent, however, the user's player sends back another set of available bandwidth information to the server, which then chooses whether to continue sending chunks from the current file or to switch to a file of a different bandwidth. This decision-making process happens multiple times throughout a streaming session and can accommodate network fluctuations that both lower and raise the available bandwidth.

What's the Big Deal?

So why is HTTP streaming important?

First, based on the pricing of previous solutions, the costs of HTTP streaming delivery may end up being lower than that of traditional RTP/RTSP server solutions. I say "may" for two reasons: first, this is based on historic pricing structures where some CDNs have offered basic HTTP progressive download delivery services, without calling them HTTP streaming, and have charged less for this service than true streaming delivery; second, HTTP servers are extremely cost-effective, and there is the ability to scale HTTP servers at a lower price than traditional streaming servers.

The second reason HTTP streaming is important, as previously mentioned, is that true HTTP streaming provides the ability to push video streaming content through port 80, which is open on many firewalls in order to allow traditional web traffic that is delivered via HTTP, while still retaining the ability to throttle back the content to a data rate just above that of the player.

For instance, Microsoft's HTTP Streaming Protocol "is suitable for streaming delivery at a fixed rate or at a data rate correlating closely to [a] rate at which the video will be displayed by the receiver."

Third, HTTP streaming allows on-demand media to be more easily delivered to caches along the edge of the network, closer to the end users.

Fourth, and perhaps the most important for our discussion, is the fact that HTTP streaming addresses both buffering and network anomalies, enhancing the viewership experience while also limiting the number of wasted bits. HTTP streaming can not only throttle back delivery, as I mentioned, but it also can allow even higher-quality content to be streamed, should a user's network conditions improve.

Finally, like other streaming protocols, HTTP streaming allows back-channel communications between

SEARCH STREAMINGMEDIA.COM



UPCOMING INDUSTRY CONFERENCES
Streaming Forum 2014 (London) [24-25 June 2014]
Streaming Media West 2014 [18-19 November 2014]
Streaming Media Producer Live [18-19 November 2014]
Recent Conferences
Streaming Media East 2014 [13-14 May 2014]
Content Delivery Summit 2014 [12 May 2014]
Streaming Media West 2013 [19-20 November 2013]
Streaming Media Producer Live 2013 [19-20 November 2013]
Past Conferences

WOWZA media systems
+
amazon web services™

POPULAR ARTICLES NEWS FORUMS
TOP 10 MOST POPULAR ARTICLES
How to Create Interactive HTML5 Video
eBay Embraces Enterprise YouTube
WD Intros WD TV, Its First Set-Top Box with Miracast Support
The ROI of Enterprise Webcasting: Justifying the Expense
MPEG LA's HEVC Licensing Terms Are Flawed, Will Prevent Adoption
Anatomy of a Viral Video: The First Days Are the Most Important
How Be Inspired Films Shoots

streaming media
Brought to you by:

the server and the client, allowing the client's player to send feedback to the server requesting a change to the transmission rate—either at a higher or a lower bitrate—to a secondary stream, referred to as "intelligent streaming" or "adaptive streaming."

Since this requesting is done, however, in a tightly controlled time frame, the added benefit is that the user is not downloading significant amounts of content that will later be abandoned. In addition, on Windows Media files, HTTP streaming can also be used to send commands such as fast-forward, rewind, pause, and location seek as trick mode requests, traditionally identified as streaming-only functionality.

In other words, the combination of streaming's benefits coupled with the consistency of progressive download means that HTTP streaming can deliver content without long buffering or waiting periods and is less susceptible to network hiccups than in traditional or multibitrate streaming.

« previous **Page 4 of 5** next »

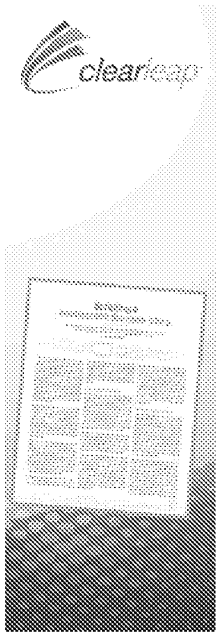
Post a comment...

Comment using...

Facebook social plugin

Related Articles

Online Video News: Encoding.com Now Supports Apple HTTP Streaming and the iPad
Cloud-based encoding service Encoding.com has added HTTP Live Streaming, with presets for iPhone, iPod Touch, and now the iPad.





***Streaming Media* Magazine**

June 2014
Subscribe

Current Issue **Past Issues**

- How to Create Interactive HTML5 Video
- How to Develop a Viable 4K Production Workflow
- How Be Inspired Films Shoots Conferences the TED Way
- Twitch Is a Billion Dollar Surprise, Even for Video Insiders
- Don't Just Compress the Video, Compress the Content

Current Digital Edition



SPONSORED LINKS

Looking for Trusted Suppliers? Check out the Industry Directory here

INSIDE STREAMINGMEDIA.COM

Blog
SM Magazine
SM Digital Edition
Streaming Media App
Streaming Media TV
Conference Videos
Video Tutorials
Readers Choice
Research Reports
Press Releases
Web Events
About Us
Newsletters
Industry News
SM White Papers
Industry White Papers
Sponsored Articles
Forums
Discussion Lists
Industry Directory
Industry Jobs
Contact Us
What is Streaming

RESEARCH CENTERS

TOPICS
Mobile Video
Music/Radio
Online Video Platforms
OTT Video
Video Advertising
Video Content Protection/DRM
Video Creation/Production
Video Encoding/Transcoding
Video Formats/Players/Codecs
Video Hosting/Delivery
Video Legal Issues
Video SEO

VERTICALS
Education Video
Enterprise Video
Government Video
Media/Entertainment Video
Online Video Industry

OTHER IT/ SITES

Streaming Media Europe
Streaming Media Producer
destinationCRM
ITContent
Faulkner Information Services
Information Today
Information Today Europe
ITResearch.com
KMWorld
OnlineVideo.net
Smart Customer Service
Speech Technology

© 2000 - 2014, StreamingMedia.com, an Information Today, Inc. (ITI) company

[PRIVACY/COOKIES POLICY]



HTTP Streaming: What You Need to Know

Understanding this hybrid delivery approach is key to advancing robust streaming solutions.

By *Tim Siglin*

For the rest of the February/March 2010 (Sourcebook) issue of Streaming Media magazine please *click here*

« previous **Page 5 of 5**

SOCIALBOOKMARK

Email
Print
Digg
Send
Like 1
Tweet 1
Share 0
+1 0

Companies Mentioned:

Adobe
Akamai
Microsoft
octoshape

Forget, Relearn, Repeat

If you are considering HTTP streaming coupled with adaptive bitrate file chunking to produce HTTP adaptive bitrate streams, what do you need to forget or relearn?

First, forget what you know about keyframes. Typically, a 5–7 second keyframe "rule of thumb" has been in play as a balance between too many and too few keyframes. Most adaptive bitrate solutions have been optimized to have chunks of less than 4 seconds, meaning that a keyframe will need to occur within each chunk.

Second, learn that you likely will have to re-encode content. The need to simultaneously segment the same content across multiple files of varying bitrates means that it is important to use the same segmenting algorithms. Even those companies that state that re-encoding is not needed will also say that audio keyframing must match up between all the files and that audio keyframes must be no longer than the chunk duration—otherwise there will be audio gaps and inconsistency in audio playback.

Third, learn that the GOP is your friend. No, not the Grand Old Party, unless you're of a particular political bent; rather the Group of Pictures (or Long GOP if you're familiar with MPEG-2 terminology). Most adaptive bitrate solutions look at groups of pictures (GOPs) and place segment points at the beginning of each GOP; the beginning of each GOP is a full I-frame rather than a P- or B-frame.

Fourth, forget ASF. A staple of Microsoft streaming media for years, the ASF format has been replaced in Smooth Streaming by the MP4 file extension, also known as the ISO/IEC 14496-12 ISO base media file format specification. The MPEG-4 system, which uses the MP4 extension, is based on an early QuickTime player format. According to Microsoft, MP4 has less overhead than ASF and is easier to parse in managed (.NET) code than ASF.

Fifth, learn that MP4 is set to become the key file extension for adaptive bitrate video files. Apple and Adobe have used the MP4 extension for some time as its container structure supports the H.264 video codec and advanced audio codec. While ASF can also contain H.264 video, it's not as straightforward an implementation for H.264, according to Microsoft. In addition, the MP4 container can also handle other audio and video codecs, such as Microsoft's VC-1 (a SMPTE standard) and Windows Media Audio (.wma) codecs.

MP4 was designed to natively support what is known as "payload fragmentation" within the file. Given the fragmentation, or chunking, of adaptive bitrate streaming, the MP4 format can be internally organized as a series of fragments.

"MP4 boxes [may be] organized in a fragmented manner," said Microsoft's Alex Zambelli in a blog post on Smooth Streaming's implementation of the MP4 system. "[An MP4] can be written … as a series of short metadata/ data box pairs, rather than one long metadata/data pair. The Smooth Streaming file format heavily leverages this aspect of the MP4 file specification, to the point where at Microsoft we often interchangeably refer to Smooth Streaming files as 'Fragmented MP4 files. …'"

"We say that the Smooth Streaming format is based on the MP4 file format," continued Zambelli, "because even though we're following the ISO specification, we specify our own box organization schema and some custom boxes. In order to differentiate Smooth Streaming files from 'vanilla' MP4 files, we use new file extensions: *.ismv (video+audio) and *.isma (audio only)."

What's Next? Multicasting and P2P

SEARCH STREAMINGMEDIA.COM



UPCOMING INDUSTRY CONFERENCES
Streaming Forum 2014 (London) [24-25 June 2014]
Streaming Media West 2014 [18-19 November 2014]
Streaming Media Producer Live [18-19 November 2014]
Recent Conferences
Streaming Media East 2014 [13-14 May 2014]
Content Delivery Summit 2014 [12 May 2014]
Streaming Media West 2013 [19-20 November 2013]
Streaming Media Producer Live 2013 [19-20 November 2013]
Past Conferences



POPULAR ARTICLES NEWS FORUMS

TOP 10 MOST POPULAR ARTICLES
How to Create Interactive HTML5 Video
eBay Embraces Enterprise YouTube
WD Intros WD TV, Its First Set-Top Box with Miracast Support
The ROI of Enterprise Webcasting: Justifying the Expense
MPEG LA's HEVC Licensing Terms Are Flawed, Will Prevent Adoption
Anatomy of a Viral Video: The First Days Are the Most Important
How Be Inspired Films Shoots

The use of HTTP delivery—including HTTP progressive downloads, homegrown HTTP streaming solutions used by Akamai, and adaptive bitrate solutions launched by Adobe, Apple, and Microsoft—will provide stop-gap measures when they are implemented in 2010, allowing for a larger quantity of "generic" servers to be brought to bear for large events.

Beyond these solutions, though, highly scalable solutions will require multicasting or some form of peer-to-peer delivery. Multicasting requires a coordinated agreement between router, switch, and LAN/WAN product manufacturers, as well as Tier 1 service providers. While we may eventually see multicasting, other hybrid solutions will include multicasting for the LAN and unicasting for the WAN. Examples of these peer-to-peer delivery solutions include a plug-in from Octoshape as well as Adobe's forthcoming RTMFP. Peer-to-peer solutions may help address ISP-level broadcast issues at television-sized audiences, even if the pipes from the ISP to the rest of the world are fairly narrow.

« previous **Page 5 of 5**

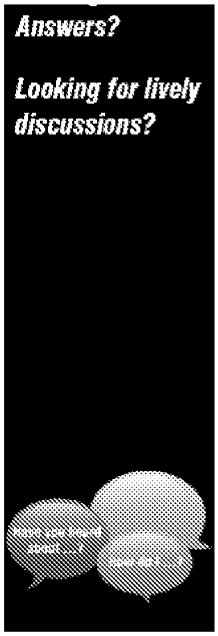
Add a comment...

Comment using...

Facebook social plugin

Related Articles

Online Video News: Encoding.com Now Supports Apple HTTP Streaming and the iPad
Cloud-based encoding service Encoding.com has added HTTP Live Streaming, with presets for iPhone, iPod Touch, and now the iPad.





***Streaming Media* Magazine**

June 2014
Subscribe

Current Issue **Past Issues**

- How to Create Interactive HTML5 Video
- How to Develop a Viable 4K Production Workflow
- How Be Inspired Films Shoots Conferences the TED Way
- Twitch Is a Billion Dollar Surprise, Even for Video Insiders
- Don't Just Compress the Video, Compress the Content

Current Digital Edition



SPONSORED LINKS

Looking for Trusted Suppliers? Check out the Industry Directory here

INSIDE STREAMINGMEDIA.COM

Blog
SM Magazine
SM Digital Edition
Streaming Media App
Streaming Media TV
Conference Videos
Video Tutorials
Readers Choice
Research Reports
Press Releases
Web Events
About Us
Newsletters
Industry News
SM White Papers
Industry White Papers
Sponsored Articles
Forums
Discussion Lists
Industry Directory
Industry Jobs
Contact Us
What is Streaming

RESEARCH CENTERS

TOPICS
Mobile Video
Music/Radio
Online Video Platforms
OTT Video
Video Advertising
Video Content Protection/DRM
Video Creation/Production
Video Encoding/Transcoding
Video Formats/Players/Codecs
Video Hosting/Delivery
Video Legal Issues
Video SEO

VERTICALS
Education Video
Enterprise Video
Government Video
Media/Entertainment Video
Online Video Industry

OTHER IT/ SITES

Streaming Media Europe
Streaming Media Producer
destinationCRM
ITContent
Faulkner Information Services
Information Today
Information Today Europe
ITResearch.com
KMWorld
OnlineVideo.net
Smart Customer Service
Speech Technology

© 2000 - 2014, StreamingMedia.com, an Information Today, Inc. (ITI) company

[PRIVACY/COOKIES POLICY]



Unifying Global Video Strategies

MP4 File Fragmentation For Broadcast, Mobile and Web Delivery

A *Transitions in Technology* White Paper

Timothy Siglin
Co-Founder
Transitions, Inc.
16 November 2011

Unifying Global Video Strategies

MP4 File Fragmentation For Broadcast, Mobile and Web Delivery

Consistent multi-platform audio and video content delivery presents an ongoing challenge for broadcasters. Explosive smartphone and tablet growth on varying operating systems —Android™, Apple® iOS, or Windows® Phone—threatens to create a user-experience divide between users on mobile devices, at the desktop or in the living room.

Broadcasters must address multi-platform consumption demands without compromising content security or network efficiencies. Many broadcasters are assessing efficiency of transport protocols used for content delivery, to see how they stack up for web and mobile delivery. Some legacy solutions, such as MPEG-2 Transport Stream (M2TS), lack basic web delivery functions.

What key information do broadcasters and network operators need to know as they look for more efficient approaches to the media delivery? This white paper explores fragmented MP4 files (fMP4) and considers whether the fMP4 format can replace legacy file formats.

Along the way, we'll explore four key areas that impact both broadcasters and network operators:

- Format benefits of fMP4
- Network benefits of fMP4
- Movement toward fMP4 standardization
- Platforms supporting fMP4

The Challenge

MPEG-2 Transport Streams (M2TS), in use for almost two decades, have been deployed for widely varied broadcast and physical media delivery solutions (e.g. ATSC and Blu-Ray). However, little has changed in that timeframe for basic M2TS transport stream capabilities. For instance M2TS still lacks an integrated solution for digital rights management (DRM). The M2TS derivative used by the proprietary Apple® HTTP Live Streaming (HLS) protocol also lacks timed-text or closed-captioning features such as CEA 708 for ATSC TV or SMPTE Timed Text for fMP4 Common File Format.

MPEG-4 versus MP4

MPEG-4 is often confused with MP4 (a particular MPEG-4 file format). The MPEG-4 standard's 28 parts detail audio, video and file structure specifications. Part 3 is for AAC audio (ISO/IEC 14496-3) and Part 10 covers AVC video (ITU-T H.264 or ISO/IEC 14496-10). Part 12 covers ISO Base Media File Format (ISOBMFF) using *boxes* (or *atoms* in the QuickTime® file format, the basis for Part 12). Part 14 defines MP4 file format with multiplexed audio/video samples. Fragmented MP4 (or fMP4) refers to fragmenting Part 12 ISOBMFF using non-multiplexed audio/video.

Appendix A shows how key boxes ('mdat', 'moof', 'moov') fit together.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1201
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Benefits of the Fragmented MP4 Approach

Is fragmenting MP4 an alternative solution that addresses the shortcomings of M2TS? It appears the answer is "yes" as fragmented MP4 files have six key benefits:

- Content/Metadata Separation
- Independent Track Storage
- Trick-Play Modes
- Backwards Compatibility
- Seamless stream splicing
- Integrated DRM

Separating Content and Metadata. With MP4 files, metadata can be stored separately from audio and video content (or "media data"). Conversely, M2TS multiplexes (combines) multiple elementary streams along with header/metadata information.

In other words, the MP4 approach allows flexible placement of metadata, independent of where media data is stored. This separation of content and metadata allows specification of a format using movie fragments optimized specifically for adaptive switching between different media data, whether that media data is varying video qualities, multiple camera angles, different language tracks or even different captioning or time-texted data.

Independent Track Storage. In fragmented MP4 (fMP4), elementary streams are not multiplexed and are independently stored separate from metadata, as noted above. As such, fMP4 file storage is location agnostic and can store audio- or video-only elementary streams in separate locations that are addressable from XML-based manifest files. One benefit of separating metadata and media data is the ease with which a stream combination can be aggregated by the playback device. An elementary audio file (m4a) stored at one location can be combined with an elementary stream video file (m4v) stored at another. The manifest file combines both together, by downloading and synchronizing their movie fragments, without requiring a multiplex of several elementary streams beforehand on the server.

The fragment-addressable nature of fMP4 stores all of an audio or video track's fragments in the same file, unlike the proprietary HLS approach which uses thousands of small M2TS multiplexed files. Audio and video segments in fMP4 can be addressed separately as single fragments of bytes, with each fragment containing the necessary metadata and media data to make it decryptable and decodable. A manifest file for fMP4 needs only a template to generate addresses for audio or video movie fragments, rather than the HLS approach of a manifest containing a list of every file name for the thousands of short file chunks needed to construct every multiplexed audio/video combination.

Trick-play modes. Trick-play modes—such as fast-forward or reverse, slow motion, and random access to chapter points—can be accomplished via intelligent use of metadata. Super fast forward and reverse are also supported by requesting only the start of each movie fragment containing the 'moof' box and the start of the 'mdat', which contains a random access picture (AVC IDR picture).

Transport stream chunks may not start with entry points, so skipping from one keyframe to another is difficult without parsing through the entire set of TS packets. In addition, fMP4 supports multiple

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1202
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

playback modes with scalable codecs like SVC that can encode different frame rates in the same media data so a player can request and play different frame rates from the fMP4 container.

The Unseen Danger of Combinatorial Complexity

The one-to-one distribution model of web video delivery works against M2TS delivery. Consider the following scenario:

A typical DVD movie in Europe or Asia requires several discs with different combinations of audio/language, subtitles, description tracks for the hearing or visually impaired, stereo, multichannel, various codecs, commentaries, camera angles and the like. All told, more than 40 different audio, video, and subtitle tracks may be distributed in a region.

Each video track needs to be encoded at several bitrates for adaptive streaming delivery at multiple resolutions for displays ranging from cell phones to HD TVs. The number of tracks can easily exceed those on a typical single-region DVD. Streaming is usually bandwidth limited, so only the tracks a user selects for playback should be delivered to a particular user, since unused audio and subtitle tracks directly reduce video quality in proportion to the bandwidth wasted. For multiplexed video formats, such as M2TS, that means a different file must be encoded for each combination of audio, video, and subtitles to be delivered.

For example, one combination could be a single video track with English audio and French subtitles, or another with French audio and English subtitles. Each combination needs to be encoded at several bitrates.

This plethora of combinations can quickly become unwieldy, with upwards of 4000 possible transport streams combinations—a multiple far above the 40 audio, video and subtitle tracks required for the DVD example. Netflix® estimates using an M2TS / HLS approach for their library could result in several billion assets, due to combinatorial complexity—a situation they avoid by using an fMP4 elementary stream approach, by selecting and combining tracks on the player.

Keeping track of these possible combinations, if encoded and multiplexed ahead of time, becomes a content management nightmare, especially since some transport stream solutions consist of thousands of small files for each transport stream combination. Even if a byte-range solution were available, however, the asset management of up to 4000 different transport streams becomes an unnecessary and cumbersome approach to single-stream delivery.

Tracks for broadcast purposes are typically limited to a primary audio, secondary audio for another language, commentary audio and subtitles in one or more alternate languages. This limited grouping of tracks, when applied to streaming delivery, still requires a large number of combinations. Take three different resolutions for a variety of screen sizes, add eighteen video tracks at different bitrates for desktop, mobile and tablet resolutions, then top it off with a number of audio and subtitle tracks for language and accessibility purposes. Based on this simple example, it is easy to see how even basic DVD video and audio delivery on the web can quickly generate many multiplexed files due to combinatorial complexity, and why the fMP4 approach greatly simplifies delivery

Backwards compatibility to MPEG-2 Transport Streams. Separating timing information from elementary video and audio streams allows conversion from several fragmented MP4 elementary streams into a multiplexed M2TS. For Adobe® and Microsoft®, time-alignment properties inherent to fMP4 means their respective server solutions can easily convert to the proprietary Apple HLS protocol for iPhone® and iPad® devices, which uses MPEG-2 Transport Stream segments.

“There are many media formats in the world,” says Kilroy Hughes, a Microsoft senior digital media architect, “so media interoperability between services and devices is a huge problem for Internet

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1203
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

video. We feel the best solution is to migrate to a Common File Format for Internet video delivery, analogous to standardizing on DVD in packaged media. When an HLS device requests a stream, we create an M2TS on the fly from the requested combination of independent tracks stored in Common File Format, which uses the fragmented ISO Media file format. MP4 elementary streams are time-aligned, so it's easy to create MPEG-2 multiplex chunks corresponding to each movie fragment from the MPEG-4 stored video and audio elementary streams."

One of the reasons this conversion from MP4 elementary streams to HLS (M2TS) is possible is the way that MPEG-4 stores streams. MPEG-4 houses raw elementary streams with integrated DRM using the Common Encryption format, while M2TS stores in the older multiplexed Annex B format that limits playback compatibility. Even if M2TS were used to create separate audio- and video-only transport stream, it is still difficult to recombine them using unrelated encoding timestamps stored in an M2TS stream.

Seamless stream splicing. Fragmented MP4 audio or video elementary streams can be requested separately because each movie fragment only contains one media type. By contrast, given M2TS's multiplexed nature, a chunk normally contains a mix of media types that can't be requested separately. This means that, even if an M2TS could be accessed at a byte-addressable level, the requested byte-range will be a mash-up of metadata, audio and video content, and header information. Any attempt to download multiple multiplexed files and select tracks on the player requires twice the download bandwidth, buffering, and demultiplex processing on the player, and has additional synchronization problems aligning tracks from different transport streams, and splicing them during adaptive switching.

"From the early days of adaptive streaming," said Hughes, "attempts to do bitstream splicing using M2TS relied on strict encoding to allow concatenating one chunk with another chunk to make a decodable stream. Concatenating different audio or packet ordering from two different files results in undecodable streams. When M2TS chunks aren't time aligned or don't start with IDR frames, duplicate chunks must be downloaded in the same time interval and approximate splices made at different audio and video locations—at significant network efficiency and device complexity costs."

Fragmented MP4 sync points are matched by bitrate and media type: audio sync points should all match each other as should video sync points, but sync points for different media types don't need to match each other. Accurate bitstream splices require identical timing for proper audio-video sync between two streams. Time-aligning audio and video elementary streams separately means sync points can be perfectly aligned to Groups of Pictures (GOPs) in video or sync frames in audio.

Using independent sync points for media types along with simple concatenation splicing yields shortened segment lengths: M2TS segment lengths are typically ten seconds, limiting frequency of switching and require extra bandwidth and processing. Segment lengths for fMP4 can be as low as 1 or 2 seconds as they don't consume more bandwidth or require extra processing. Short

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1204
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

segments make quicker adjustments in bitrate to avoid dramatic video quality changes or rebuffer pauses, and more time-accurate splice points for ad insertion.

One challenge for DVD-like Internet delivery is alternate camera angles. Time aligning alternate angles, though, allows adaptive streaming to duplicate the seamless switching of a DVD.

Overlap Splicing: Device Limitations

Overlap splicing is required in adaptive streaming formats such as HLS, when alternate files do not have time aligned switch points that allow requested file chunks from alternate bitrate files to be simply concatenated and run through a single demultiplexer and decoders for audio, video, and subtitles. Overlap splicing requires multiple overlapping file chunks from new and old bitrates to be downloaded and processed in parallel until an entry point in the new video stream is reached in the decode and display sequence and the remaining chunk of the previous bitrate can be discarded. This requires additional buffering, processing, and analysis of the timestamps of two transport streams to conform the contained packets to a common timeline.

The additional memory and processing required for overlap splicing can negatively affect audio/video quality and battery life in a device, but its most serious cost may be network performance. Downloading two file chunks in the same time period and discarding the first portion of the new bitrate chunk and the last portion of the old bitrate chunk roughly cuts network efficiency in half when adaptation is taking place.

This is a huge performance problem in the typical case when switching is required to prevent input buffer underflow and a "Paused for rebuffering" error message. In order to switch to a lower bitrate to prevent underflow, a device must download a file chunk at the existing bitrate as well as a file chunk at the new bitrate in order to maintain playback of the old bitrate while processing the new file chunk to find audio and video splice points. Since the network is unable to keep the buffer full downloading one file chunk per time interval, it increases the risk of a buffer underflow error to require two file chunks to be downloaded in the same timer interval in order to change bitrates.

A format such as fMP4 reduces network bandwidth immediately when switching by requesting the next video fragment at a lower bitrate instead of (not in addition to) the higher bitrate fragment next in sequence. Fragments are concatenated and decoded without any additional processing requirement. Separation of audio, video, and subtitle tracks means each track can be chunked exactly at their access points, and concatenated without disturbing the tracks timeline or decoding.

Integrated Digital Rights Management (DRM). Inherent to the MPEG-4 specification, digital rights and—if necessary—encryption, can both be applied at the packet level. In addition, the emerging MPEG Common Encryption (ISO/IEC 23001-7 CENC) scheme can be combined with MPEG-4 Part 12 to enable DRM and elementary stream encryption.

AVC elementary stream encryption is applied to the NAL (Network Abstraction Layer) unit. Streams can be edited without decryption, enhancing compatibility with different decoders and allowing re-containment (moving encrypted elementary streams from several fMP4 files to a single M2TS file).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1205
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

With a number of available DRM schemes, however, it is not practical to create a different version of each file for each DRM scheme. MPEG Common Encryption (CENC) scheme shares a multi-DRM scheme approved for UltraViolet (UVVU) that uses five DRM solutions—one each from Adobe, Google, Marlin, Microsoft and the Open Mobile Alliance. Work has been done to make CENC an MPEG standard as well, for interoperability across a number of devices and DRMs. Both Adobe Flash® Access and Microsoft Protected Interoperable File Format (PIFF) reference CENC.

MPEG-DASH players share CENC for elementary streams, although pure M2TS lacks common encryption due to multiplexing and a lack of indexed metadata separate from elementary streams.

Combining CENC to the Common File Format (CFF) championed by the Digital Entertainment Content Ecosystem (DECE) may yield some interesting results. When CENC is ratified shortly as MPEG Part 12 Amendment 3, proponents say CFF and CENC will represent two important steps toward large-scale online video distribution via adaptive delivery of fragmented elementary streams.

Since CFF can also be used outside of the bounds of UltraViolet, significant interoperability may also exist between UltraViolet disc-based playback and online video platforms, in much the same way that the DVD Forum's published specifications for DVD playback guaranteed interoperability between DVD players. It's not out of bounds to think of CFF as the DVD standard for the web.

Network Effects of fMP4 versus M2TS

Beyond the basic functional differences of fragmented MP4 and MPEG-2 Transport Streams, it is worth exploring the effects of each, from the perspective of bandwidth and cache efficiency.

Caching efficiencies. When considering comparative complexity, noted in the sidebar above, it's also worth considering the effect that all the possible combinations has on cache efficiency. Each track and bitrate combination required for M2TS delivery means a higher likelihood that an edge cache will not have the particular combination, resulting in a request back to the origin server.

With fMP4, cache-hit ratios are improved almost by a factor of 100. Since fMP4 manifest files use the same fMP4 fragments from a single elementary stream combined on the device, constant use of the same fragments across multiple combination requests works to increase the cache-hit ratio. The likelihood is higher, then, that a fragment will reside on a nearby edge cache (e.g. same audio fragment used for all video bitrates, resolutions, and camera angles). In our DVD example, the 40 fMP4 elementary streams could populate a nearby edge cache, but many of the 4000 M2TS combinations would reside on the origin server until a particular combination request is made.

Bandwidth efficiencies in segment alignment. A key factor when generating multiple files at different bitrates is segment alignment. HTTP adaptive streams via fMP4 relies on movie fragments each being aligned to exactly the same keyframe, whether for on-demand or live streaming (live

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1206
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

requires simultaneous encoding of all bitrates to guarantee time alignment). A packaging tool that is fragment-alignment aware maintains elementary stream alignment for previously recorded content.

Time alignment and media synchronization to keyframes is equally critical for two-pass encoding. As a best practice, the first-pass output should be used for all second-pass and multibitrate encodings to assure keyframe alignment. But what if time-alignment information is arbitrary?

Arbitrary time alignment creates both device and network efficiency issues. If a device encounters bandwidth deficiency and requests a segment at a different bitrate, arbitrary time alignment means the device will be sent two segments (twice as much data) when a device requests less data. Demultiplexing and splicing disparate multiplexed transport streams into a desired combination of audio-video-text content also impacts device processing power and battery life (sidebar, page 6).

Proprietary solutions like Apple HLS allow arbitrary I-frame alignment but often require downloading two segments at a time, which is detrimental to network efficiency. When bandwidth fluctuates, HLS requires almost twice the amount of bandwidth in order to switch between the current and next-available bitrate available in the manifest.

Let's assume a user has 1 mbps of available bandwidth to view content. If no bandwidth variation occurs, a user would view the entire content piece at 1 mbps. When bandwidth fluctuates, though, a user's player then needs to request a different bitrate to watch the remaining content. A switch needs to occur between the current 1 mbps stream and the next-closest bitrate in the manifest.

In order to accomplish overlap switching, two segments of considerable length—Apple HLS typically uses10-second segments—are simultaneously downloaded to begin overlap splicing. More information about overlap splicing can be found in the sidebar, but the point about network efficiencies should not be lost in the debate between concatenation and overlap approaches: short of the necessity to download two segments for overlap splicing, there is no reason to tax the network in this way at the very moment at which bandwidth has dropped considerably. Yet this is precisely what occurs in the overlap splicing model.

How does an fMP4 approach stack up? Latency goes down and ongoing visual quality goes up.

With shorter segment lengths come lower latencies, a boon for live streaming and one Microsoft exploited in streaming the last few Olympic Games. According to numerous sources, a good balance between efficiency and switching latency is to insert keyframes every 2-4 seconds.

Second, the perception of ongoing quality directly correlates to the length of any given fragment. Seamless switching allows for use of 2-second segments, where the complexity of overlap splicing means an M2TS approach has a typical 10-second segment length. A 2-second increment has a much less visible image quality shift between segments than a 10-second increment. The end result of shorter segments is a lower perceived quality variation between A-B visual comparisons.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1207
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The Move Toward Standardization of fMP4

A protocol is only as good as its adoption rate, both in terms of interoperability and critical mass. Many standards, including mobile, MPEG-DASH and UltraViolet (UVVU) standards, are using fMP4.

MPEG-DASH. MPEG, the standards body responsible for MPEG-2 and MPEG-4, is addressing dynamic adaptive streaming over HTTP (MPEG-DASH) through the use of four key profiles—two around CFF for fMP4 and two for MPEG-2 TS. For fMP4 both the live (ISO LIVE) and on-demand (ISO ON DEMAND) profiles reference Common Encryption (CENC or ISO 23001-7). ISO LIVE, a superset of live and on-demand profiles, is close to Microsoft Smooth Streaming, meaning Microsoft could easily update its server and client viewers to meet DASH compliance.

For MPEG-2 TS, there are two profiles: Simple and Main. Simple uses constrained encoding and decoding parameters to comply with M2TS chips in web-connected DVD players or set-top boxes.

The Transport Stream Main profile allows for unconstrained encoders and arbitrary segment lengths that may not begin with an I-frame, similar to the Pantos specification. The "Pantos spec" is Apple's submission to the Internet Engineering Task Force (IETF) for an HTTP streaming format. However, according to an MPEG-DASH panelist at the 2011 Streaming Media West show, the Pantos spec introduced to IETF is merely an informational draft, not a specific standards request.

While DASH-compliant players expect a DASH-compatible manifest—meaning that it's not possible to use an Apple .m3u8 playlist—it's easy to see that a DASH player could handle both standards-compliant M2TS and fMP4 decoding in a single player. Transport Stream Main Profile allows an XML-based manifest (an MPD) to serve M2TS fragments prepared for initial use in HLS. The TS Main profile in DASH offers Apple a path toward standards compliance and interoperability.

Many companies are considering implementation of the Common File Format profiles. One that's been in the news quite a bit recently, Netflix, stated its reasons for supporting MPEG-DASH:

> *The proposed DASH standard defines a way to advertise a range of different streams to a player together with the information it needs to pick which ones to stream. It also defines media file formats suitable for adaptive streaming. The file formats enable efficient and seamless switching between streams, enabling a player to adapt to changing network conditions without pausing playback for re-buffering. The standard considers the differing needs of both on-demand services such as ours, and live services. And it's all based on the use of industry standard HTTP servers.*

DASH's "range of different streams" is also known an Adaptation Set. Representations are listed in an MPD (Media Presentation Description), each with a specified bitrate. A device picks the most likely bitrate for its network, then adjusts bitrates by changing the Representation from which it selects Segments as its buffer either gets too full or to empty based on actual network throughput.

An Adaptation Set may contain the same bitrates for cell and Wi-Fi networks (e.g. 300, 450 kbps).

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1208
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

As with other adaptive bitrate considerations, audio and aspect ratio are key: audio streams should be exactly the same, and while video can be different dimensions and bitrates, the video aspect ratio and frame rates should remain constant between video streams.

Some solutions use a command-line authoring tool to create a variant playlist, based on previously chosen media segment parameters, while others integrate the variant playlist creation directly into the media segmentation workflow. It should also be noted that, while variant or multiple playlists are supported by a number of servers, the client end must initiate a call for the correct playlist. In much the same way that CSS3 is used for browser queries to determine whether the page to be served up is on a desktop, mobile or tablet browser, the server relies on the client player to request the proper variant playlist, depending on whether the player senses a cellular or Wi-Fi connection.

DASH also allows identification of tracks that are entirely random access pictures so they can be scanned backwards and forwards to find the desired spot to resume playback.

UltraViolet. This digital-rights management and delivery standard—using key DRM (digital rights management) schemes from Adobe, Google, Marlin, Microsoft and the Open Mobile Alliance—allows content owners to deliver in a common file format with the Common Encryption standard.

The 70-member Digital Entertainment Content Ecosystem (DECE) and six studios back UltraViolet, which uses ISO Base File Format to ensure codec, DRM, timed text and media format consistency.

This common file format (CFF) in UltraViolet is itself based on Microsoft Protected Interoperable File Format (PIFF) and PIFF 1.3 is compatible at the container layer with CFF. A common file format allows interoperable playback and file copying across a number of consumer electronics (CE) devices and computers as the CFF end goal. CFF files can be played on software- or hardware-based UltraViolet players, using any one of the five CENC common DRM systems via an AES cipher and key identifiers embedded within the fMP4 fragments ('moof').

All three profiles—portable definition (PD), standard definition (SD), and high definition (HD)—use the fMP4 container format (ISO/IEC 14496-12). This ISO container uses H.264/AVC video (ISO/IEC 14496-10), allowing multiple resolutions, aspect ratios, and frame rates for progressive-scan video.

UltraViolet leverages fMP4 to allow for SMPTE Timed Text (SMPTE TT), incorporating Unicode text and PNG graphics for captions/subtitles for the hearing-impaired user (SDH). Sign language, dialog translation, and commentaries can also be displayed through the use of subtitles and subpictures.

Fragments can reside at a CDN or in "the cloud" for easy access by consumers. UltraViolet uses a "digital locker" containing digital rights information for content, but not content itself. UltraViolet's "coordinator" in the cloud stores rights for content purchased or rented by any member of a family but does not store content in the cloud. Retailers who sell UltraViolet content make CFF-compliant UltraViolet video files available for download, and they may be freely copied by consumers because playback is controlled by DRM licenses. UltraViolet purchase rights also entitle users to stream the titles they've purchased, and we can anticipate DASH streaming will be applied to CFF files using the DASH Live and On Demand Profiles designed for that type of fMP4 file format.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1209
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Learnings on the Path to fMP4

Major industry players have embraced the fragmented MPEG4 protocol for delivery. Some keep the ISO file formats unaltered, others have chosen proprietary extensions and maintain parallel proprietary functionality.

Two companies that contributed to this white paper—Adobe and Microsoft—are worth noting, not just for their decades-long legacy in creating compelling workflow solutions for their proprietary formats, but also for their journey towards fMP4. Their recent experience with adaptive streaming of fragmented MP4 serves both their users and the standards committees well. For insight into Adobe and Microsoft's learnings on the path to fragmented MP4, see Appendix B.

This doesn't mean either company will necessarily abandon proprietary solutions, as it is easier to enhance delivery platforms that one owns versus one that is standards-based, but both companies are committed to continuing to contribute to emerging industry standards.

Conclusion

M2TS has served its purpose, one-to-many broadcasting, for a number of years. The one-to-one nature of streaming delivery to a variety of platforms—mobile, set-top box and web—calls for a new approach to the functional and network limitations inherent to M2TS.

Fragmented MP4 (fMP4) addresses delivery variability on current mobile or fixed-line networks, providing more than just another HTTP adaptive delivery approach. With fMP4, broadcasters get real benefits: asset management of premium content, protection of assets and consistency of user experience across the growing variety of Internet connected video devices.

Seamless switching between bandwidths, without the need to encode, store, and cache segments for thousands of possible combinations, means that fMP4 can independently select and combine segments of different audio, video, and text tracks from a manifest file in order to target specific device classes or network conditions.

In addition, the re-use of elementary stream segments across multiple track combinations means caching efficiencies for HTTP delivery, leveraging better cache-hit ratios on edge servers while decreasing reliance on origin servers to deliver a consistent flow of segments to end users.

To best utilize an fMP4 workflow, a strong need for standardization exists, to move from proprietary to common file and encryption approaches. Support for the Common File Format (CFF) and Common Encryption (CENC) ensure initial interoperability across multiple online and offline use cases, while the MPEG-DASH profiles and manifests add an additional level of interoperability.

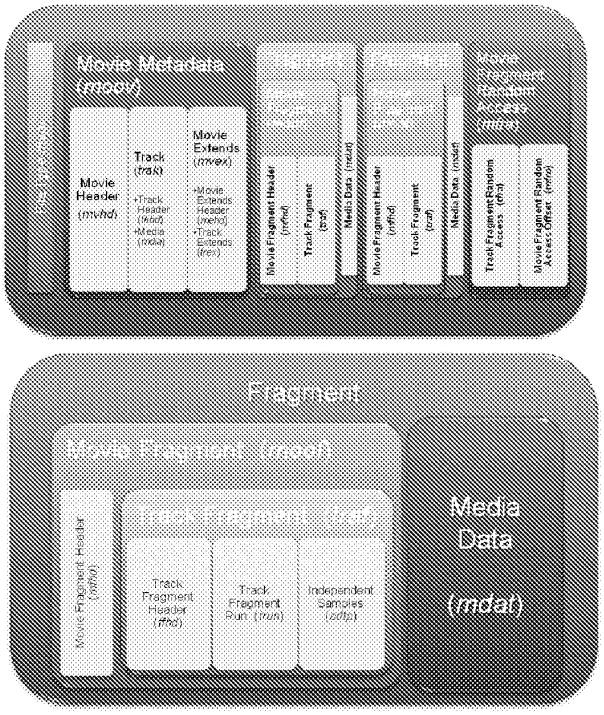
We expect to see companies like Adobe, Microsoft, Netflix and others advocate for interoperable common approaches to mutli-device delivery for consumer/professional content delivery platforms.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1210
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Appendix A: Fragmented MP4 Boxes

A fragmented MP4 file contains a number of discrete units called boxes (also known as atoms). Three key boxes in the fragmented MP4 approach are the 'moov', 'moof' and 'mdat' boxes:

- 'moov' (movie metadata box) contains a virtual movie timeline with display characteristics, duration and timescale elements, as well as sub-boxes providing information for each movie track.
- 'mdat' (media data box) contains media data, raw audio and video information and timed-text elements that are decoded based on a movie box's information.
- 'moof' (movie fragment box) contains short audio, video or text portions of an elementary stream. Movie fragments typically house a closed GOP (Group of Pictures) or AVC Coded Video Sequence.



MPEG-4 file sources did not originally support MP4 file streaming but MPEG is now exploring DASH (Dynamic Adaptive Streaming over HTTP) using MP4 elementary stream fragments as network packets for HTTP streaming delivery.

Coupling an 'mdat' and a 'moof' creates an fMP4 movie fragment that includes one track fragment (either audio or video) with sufficient metadata to make it independently decodable. Using movie fragments to create packets optimized for adaptive streaming is similar in concept to RTP packets of audio/video, but fMP4 movie fragments are optimized for bitrate switching and HTTP delivery over unreliable networks.

The two images above—based on initial Microsoft Smooth Streaming fragmentation—showcase fMP4 integration of 'moof', 'moov' and 'mdat' boxes. The images come from Alex Zambelli's blog post which offers more details.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1211
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Appendix B: Adobe & Microsoft Thoughts on Adoption of fMP4

Two companies that contributed to this white paper—Adobe and Microsoft—shared thoughts on their journey towards fMP4.

Adobe. Ask members of the Adobe Flash Media Server team what they do, and you'll likely get this response: *Adobe is in the business of delivering video experiences. Period.*

For more than a decade, that business centered on the Real-Time Media Protocol (RTMP) that Adobe launched back in 2001 with Flash Player 6. RTMP stood at the heart of Adobe Flash Media Server (FMS) and many de facto workflows sprung up around the protocol, although it was extended over the years to address encryption (RTMP-E) and peer-to-peer delivery (RTMFP).

One such enhancement to FMS was a precursor to Adobe's foray into HTTP delivery. Back in December 2009, with the release of FMS 3.5.3, Adobe essentially decoupled the buffer from the connection, allowing semi-stateless connection between player and server and empowering Flash Player to continue play back even if the connection dropped. Developers can use ActionScript to reconnect to Flash Player to FMS and, if reconnection occurs before the buffer empties, there is no perceived disruption in client playback, critical for mobile video delivery on intermittent networks.

Adobe saw opportunity to provide similar workflows for devices that did not use its Flash Player plug-in. In 2010, Adobe announced Adobe HTTP Dynamic Streaming (HDS), an HTTP-based adaptive streaming approach that built on fMP4 as its base protocol.

"This is a very different approach than our RTMP delivery with Adobe FMS, which uses normal FLV or MP4 (F4V) file formats," said Kevin Towes, Adobe's senior product manager for delivery. "The MP4-Fragment format [Adobe calls it F4F] adheres to the industry standard which is important for re-usability. The format also enabled a major requirement for media streaming – protection—allowing you to deliver your live or pre-recorded content, leverage the caching devices, and still maintain control over your video assets."

As part of the new Flash Media Server, announced in September 2011, Adobe enhanced support for both manifest (F4M) and fragment (F4F) file formats. Adobe uses an XML-based solution it calls F4M version 2, similar in concept to a variant playlist in what Adobe calls "set level manifests".

"Workflows in RTMP have more than a decade of robustness behind them," said Towes, "but the same workflows haven't been as easy in HTTP. Our investment into the HTTP workflows—bringing them on par with RTMP workflows—allows established FMS customers to maintain their sizable desktop/laptop viewer base while also pursuing new iPad and other iOS device revenue streams."

Modifications to F4F allow real-time packaging, protection and features such as variant playlists to act on par with RTMP-based workflows. For encryption, FMS encrypts F4F fragments within FMS, using either a built-in Flash Access encryption module or a stand-alone Flash Access server.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1212
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Adobe emphasizes the point that fMP4 natively allows for network optimization and content encryption, but it also notes that different devices require different optimizing tweaks for delivery. Leveraging FMS for both desktop and mobile devices, including creation of HLS-based transport streams, Adobe feels it can supply content to every device at optimal resolutions and bandwidths.

Seeking is one such area: given the ability to identify and deliver a byterange via fMP4, Adobe can bring its “seek” enhancement from RTMP directly into an HTTP delivery scenario. When enabled, seeking occurs first within the buffer, significantly reducing the load on the server compared to traditional back-and-forth seeks between client and server. The duration of the buffer can be dynamically changed, something not currently possibly in MPEG-2 TS solutions, to allow for pseudo-DVR functions such as instant replay or simple time-shifting options within the player. This also allows playback at different speeds and frame-accurate stepping from scene to scene.

“We try to help broadcasters realize new revenue streams,” said Adobe’s Kevin Towes, “regardless of platform being delivering to. Where Flash Player is available, we'll use RTMP or HTTP Dynamic Streaming to deliver world-class quality of experience, encryption and rights management. With other platforms, such as iOS-based devices, we'll do an equally good job. Adobe also sees the benefit of standards-based approaches like the Common File Format, HTML5 and MPEG-DASH.”

Microsoft. Microsoft had a significant installed Windows Media® user base for well over ten years. Yet the company announced in late 2008 it was moving toward fragmented MP4 with the launch of Smooth Streaming, a first-to-market adoption of fMP4. Smooth Streaming combines Windows Server® and Silverlight® (or other Smooth Streaming clients) with XML and SMIL manifests.

Manifests have an *.ismv extension. Manifests indicate to an adaptive streaming client what bitrates are available in a file set, where fragments reside, and header parameters from each video file.

Microsoft stores like-bitrate "chunks" of video as movie fragments in a single file, using timecode information stored in each movie fragment (moof) as an index for particular segments to download and play back. Smooth Streaming devices download movie fragments containing either a video closed Group of Pictures (GOP) or an integral number of audio sync frames, via a single HTTP request. The player than select segments (typically 2-4 seconds long) from multiple streams encoded with different bitrates, resolutions, languages, etc.

What were Microsoft’s reasons for choosing fMP4 over ASF? The company notes five:

- MP4 is a lightweight container format with less overhead than ASF
- MP4 is easier to parse in managed code (.NET) than ASF
- MP4 is itself a widely used standard, making 3rd party adoption / support more straightforward
- MP4 was architected with H.264 video codec support in mind
- MP4 was designed to natively support payload fragmentation within the file

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1213
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

“The MPEG-4 fragmented format was able to do everything we needed,” wrote Ben Waggoner, principle program manager at Microsoft, “so it was simplest to just use that rather than make up something new. We're not trying to make up a new file format here; just take advantage of existing technologies in a novel way.”

“ASF was a great format for bit-pumping,” wrote Waggoner in 2009, “with a MIPS-per-Mbps ratio better than other formats of its era. However, it simply wasn't well set up to allow a byterange to encapsulate a fragment that could potentially be a single closed GOP. Since server hardware is so much more powerful these days and proxy caching so dramatically reduces the load on the origin server, going for a somewhat more complex server-side parsing methodology made good sense.”

Microsoft chose to go “all in” around the standard, including use of *.ismv and *.mp4 extensions.

What have they done with Smooth Streaming since its announcement in 2008?

Quite a lot, actually. Smooth Streaming has seen significant traction with Silverlight, but doesn’t require Silverlight to deliver adaptive bitrate HTTP streaming, as evidenced by Smooth Streaming clients such as the Comcast® XFINITY® TV App for iOS devices and Linux-based Netgem set-top boxes. Such non-Silverlight clients can be created using a Smooth Streaming client source-porting kit from Microsoft that enables playback on any device.

In addition, publication of a Smooth Streaming specification, called PIFF (protected interoperable file format) has been adopted—along with Microsoft PlayReady® DRM technology—for use in new Netflix ready devices and applications, including the Apple iPad.

Microsoft has also enabled several compelling scenarios—including advertising and alternate language tracks—on top of fMP4. In addition, the Microsoft ecosystem surrounding the Smooth Streaming format includes two key tools: Expression® Encoder for live- and on-demand Smooth Streaming encoding and IIS Media Services. The latter rides on top of the Microsoft Internet Information Services (IIS) web server that ships in every version of Windows Server 2008.

Another very interesting part of the workflow, that I’ve covered for several trade publications as well as the Workflowed blog, is IIS Transform Manager, a server-side transcoding and transforming tool. Transform Manager can convert a variety of formats, including segmented .mp4 files, to Smooth Streaming presentations. It can also convert Smooth Streaming content for use on Apple iOS devices, converting from fMP4 to HLS (MPEG-2 TS) as part of its standard workflow. Transform Manager scales out on multi-core HPC clusters, plus allows audits of all file copies/movement to improve monitoring/management control.

Microsoft is committed to supporting the standards-based approach, including the Common File Format (CFF), Common Encryption (CENC) and the MPEG-DASH adaptive delivery over HTTP.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1214
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Attributions

Adobe Flash, Flash Media Server, Flash Player, Flash Access and PASS are registered trademarks of Adobe Systems, Inc., in the United States / other countries. Android is a trademark of Google Inc. Apple, iPad, iPhone, and QuickTime are registered trademarks of Apple Inc. The white paper has not been authorized, sponsored, or otherwise approved by Apple Inc. Expression, PlayReady, Silverlight, Windows Media and Windows Server are registered trademarks of Microsoft Corporation in the United States / other countries. Comcast XFINITY, Netflix, Netgem and Open Mobile Alliance are trademarks of their respective companies.

Acknowledgements

Special thanks to Microsoft Corporation and Adobe Systems for providing financial resources and access to subject-matter experts. The following subject-matter experts spent time expanding on key concepts and the ever-changing nature of fragmented MP4 as well as the MPEG-DASH ratification process and use cases:

- Adobe Systems. Srinivas Manapragada, Kevin Streeter, Vishy Swaminathan, Kevin Towes
- Microsoft. John Deutscher, Kilroy Hughes, Chris Knowlton, David Sayed, John Simmons, Sudheer Sirivara
- Netflix. Mark Watson

About the Author

Tim Siglin, co-founder of Transitions, Inc., holds an MBA with emphasis in entrepreneurship. Involved with visual communications for over fifteen years, Siglin's key focus is digital media entrepreneurship, including market analysis, implementation and product launches. He has served in executive roles in marketing, strategy and technology for a number of startups. He also serves as a contributing editor for several tech publications.

About the Company

Transitions, Inc., is a technology and business development firm with extensive experience in technology design and go-to-market strategy consulting. Transitions specializing in assistance to businesses seeking to identify "transition points" that hinder growth or a return to profitability. We also offer research and consulting services to several Big 5 consulting firms and internal skunk work project coordination for Fortune 500 clients.

Repeat customers account for more than 80% of all on-going business, but Transitions also takes on select project challenges assisting startups, distressed and expanding small businesses. Based in Tennessee, Transitions' business strategy and marketing consulting clients include companies in Silicon Valley, Boston, London, Milan, Mumbai, New York and Switzerland. Ongoing Transitions' projects include established businesses and startups in the digital media, financial services and global marketing industries.

© 2011 Transitions, Inc.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1215
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

VIDEO TRANSCODING FOR FAST FORWARD/REVERSE VIDEO PLAYBACK

Yap-Peng Tan, YongQing Liang, and Jun Yu

School of Electrical & Electronic Engineering
Nanyang Technological University, Singapore

ABSTRACT

Fast-forward and fast-reverse playbacks are two common video browsing functions provided in many analog and digital video players. They help users quickly find and access video segments of interest by scanning through the content of a video at a faster than normal playback speed. In this paper we propose a video transcoding approach to realizing fast forward and reverse video playbacks by generating a new compressed video from a pre-coded video. To reduce computational requirements, we design and compare several fast algorithms for estimating the motion vectors required in transcoded video. To accommodate changes due to frame skipping for fast video playback, we also alter the group-of-pictures structure of transcoded video. In addition, subjective tests are conducted to assess the minimum video peak-signal-to-noise-ratio degradation that is perceptible to viewers at different fast playback speeds. To this end, we obtain an adaptive video transcoding method, which combines intra-coding and inter-coding with a fast motion vector re-estimation method to strike a good balance between computational complexity and transcoded video quality. Experimental results are reported in this paper to show the efficacy of the proposed method.

1. INTRODUCTION

To achieve high compression ratios, many popular video compression standards, including the MPEG-1/2/4 and H.26x series, use a motion-compensated prediction scheme to reduce the temporal redundancy of video sequences. Although efficient, this prediction scheme imposes additional constraints on how a compressed video can be restored for display, as a predicted frame cannot be decoded before any of its reference frames. Furthermore, these video standards are customized in a way that only minimum computation and/or memory resources are required when a compressed video is decoded and played back in a pre-determined frame order. Showing the video in other orders often demands additional resources. For example, playing back a compressed video in reverse order would require extra memory buffer to store the intermediate reference frames.

Fast-forward and fast-reverse playbacks are two popular video browsing functions provided in many analog and digital video players. Previous work on realizing these functions in a digital video system can be found in [1]. The method, however, requires extra frames to be delivered to and processed by the decoder at a higher frame rate, imposing a heavy processing burden on both the transmission network and the decoder. Although the frame-skipping video transcoding method proposed in [2] may be used to support fast forward video playback, it does not consider the requirements for different fast speeds and the needs to alter the group-of-pictures (GOP) structure of the transcoded video. In [3], a compressed-domain approach has also recently been proposed for reverse playback of MPEG video at the normal speed. While the approach can be directly applied to support fast reverse video playback by decoding and displaying the video at a faster speed, it would require both the transmission network and the decoder to operate at a higher frame rate. For example, if the original frame rate is 30 frames/second and the target speed-up factor is 4, the approach needs to process the video at 120 frames/second. This is not an efficient or viable solution for systems with limited computational resources. Moreover, such a high frame rate may not be necessary because human eyes have only a limited temporal frequency sensitivity [4].

In this paper, we propose to realize fast-forward and fast-reverse video playbacks through video transcoding. In particular, our method only transcodes and sends the necessary frames, probably over a bandwidth-limited network, to the decoder. For example, if the speed-up factor is 4, only one out of every four frames is transcoded and the fast video playback can be easily achieved by playing back the transcoded video at the normal speed. To reduce the computational complexity, we have also designed and compared several fast motion vector re-estimation methods for transcoding video. If some frames are skipped during the transcoding process, the GOP structure of the transcoded video can be altered according to the original GOP structure and the target speed-up factor. In addition, subjective tests have also been conducted to assess the minimum amount of peak-signal-to-noise-ratio (PSNR) degradation that is perceptible to human eyes when the video is played back at different fast speeds. To this end, an adaptive video transcoding method is obtained to realize fast video playback at various speeds by combining intra-coding and inter-coding with a fast motion vector re-estimation method. The rest of the paper is organized as follows. In Section 2, we present the proposed fast-forward/reverse playback video transcoder. Section 3 reports the results of subjective quality test on video playback at different fast speeds. More experimental results on transcoding video for fast forward/reverse playback follow in Section 4, and the paper concludes in Sections 5.

2. THE PROPOSED VIDEO TRANSCODER

We propose to realize fast forward/reverse playback of compressed video through frame-skipping video transcoding. Different from the existing methods in the literature, our proposed method i) employs an intra-frame refresh scheme to alter the GOP structure of the transcoded video, ii) transcodes only the frames that are required for video playback at each different fast speed, and iii) combines intra-coding and inter-coding with fast motion vector re-estimation to strike a good balance between the computational complexity and the transcoded video quality.

Figure 1 shows the architecture of the proposed video transcoder. It consists of a typical video decoder followed by a customized video encoder. A motion vector re-estimator is included to estimate the motion vectors for the transcoded video from the motion vectors existing in the pre-coded video. A complete motion estimation, arguably the most computationally expensive operation in an encoder, is therefore not required in the proposed video transcoder. Another advantage of the proposed video transcoder is that only the frames required at the target playback speed are included in the transcoded video. Figure 1 shows an example of the proposed fast forward/reverse video transcoding process, where frame 0 to frame 17 of a pre-coded video have been restored by the decoder.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1216
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

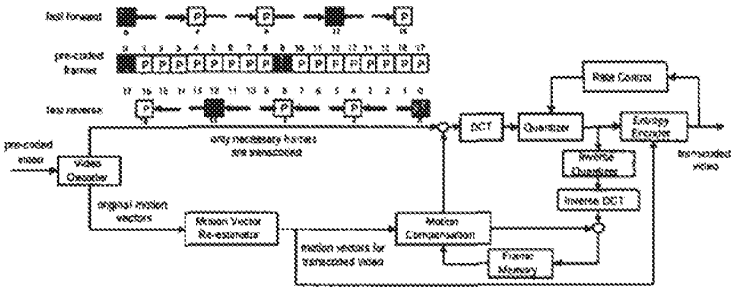


Fig. 1. The proposed video transcoder for generating fast-forward/reverse playback video.

As the target forward/reverse playback speed is 4 times the normal speed, only one out of every four pre-coded frames needs to be transcoded and sent to the target decoder. This is therefore more efficient in terms of the computation and the transmission bandwidth required. The main functions of the transcoder are further explained below.

Change of GOP Structure

Let us consider again the example shown in Figure 1. The GOP structure of the pre-coded video consists of one intra frame (I-frame) followed by eight Predicted frames (P-frames). After transcoded to video for fast forward/reverse playback, the GOP structure changes to one I-frame followed by two P-frames or one I-frame followed by one P-frame. Note that depending on the original GOP structure and the target video playback speed, the GOPs in the transcoded video may not have the same length (i.e., number of frames per GOP). In general, a frame is transcoded as an I-frame if it is the first frame selected from a new GOP in the pre-coded video, as no existing motion vectors carrying the motion information between this frame and the previous transcoded frames. Based on this observation, a simple rule can be used to determine which frame should be coded as I-frame. Let L be the length of a GOP in the pre-coded video, γ the speed-up factor for the fast forward/reserve video playback, and k the frame number of a per-coded frame to be transcoded for fast video playback. It follows that only one out of every γ frames needs to be transcoded and frame k is transcoded as an I-frame if

$$k \bmod L \quad < \quad \gamma$$

Otherwise, it is transcoded as a P-frame. It should be noted that, for the sake of simplicity, the pre-coded frames are re-numbered in reverse order when they are transcoded for fast reverse video playback, as illustrated in Figure 1.

Re-estimation of Motion Vectors

Owing to the change of GOP structure and the amount of motion existing between two successive frames in the transcoded video, new motion vectors are required for transcoding the video. To avoid a complete motion estimation using some block matching algorithm, which is usually computationally expensive, the required motion vectors can be approximately estimated from the motion vectors available in the transcoded video with much less computational requirement. Based in part on the techniques proposed for frame-skipping video transcoding [2] and the methods for estimating the reverse motion vectors of compressed video [3], we have designed and evaluated four fast motion vector re-estimation algorithms for the purpose of fast forward/reverse video transcoding, namely the "in-place", "area-weighted average", "maximum overlap", and "median" fast motion vector re-estimation methods, as detailed below.

Let $\mathrm{MB}(\mathbf{p}; k)$ be the macroblock of frame k with its top-left corner pixel located at $\mathbf{p} = (x, y)$, $\mathbf{mv}(\mathbf{p}; k)$ the motion vector of macroblock $\mathrm{MB}(\mathbf{p}; k)$ available in the pre-coded video, $\widehat{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p}; k)$ and $\widehat{\mathbf{mv}}^{R(\gamma)}(\mathbf{p}; k)$ the motion vectors estimated for macroblock $\mathrm{MB}(\mathbf{p}; k)$, with speed-up factor γ, in the fast-forward and fast-reverse transcoded videos respectively, and $A_{(\mathbf{p};k)}(\mathbf{q}; t)$ the portion (in terms of area or pixel count) of macroblock $\mathrm{MB}(\mathbf{q}; t)$ involved in forming/referencing the macroblock $\mathrm{MB}(\mathbf{p}; k)$ in the pre-coded video. Being the simplest in both concept and computation, the "in-place" method computes the motion vectors for the transcoded video as follows:

1) In-place

$$\widehat{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p}; k) = +\backslash - \sum_{t=k-\gamma+1}^{k} \mathbf{mv}(\mathbf{p}; t)$$

The other three methods ("area-weighted average", "maximum overlap", and "median") use a recursive form to estimate the required motion vectors, given as

$$\widehat{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p}; k) = \widehat{\mathbf{mv}}^{F\backslash R(\gamma-1)}(\mathbf{p}; k) + \widehat{\mathbf{mv}}^{F\backslash R(1)}(\mathbf{p} + \widehat{\mathbf{mv}}^{F\backslash R(\gamma-1)}(\mathbf{p}; k); k - \gamma + 1)$$

The main difference is in how $\widehat{\mathbf{mv}}^{F\backslash R(1)}(\mathbf{p}'; k')$ is computed, as given below for each of these three methods.

2) Area-weighted average

$$\widehat{\mathbf{mv}}^{F\backslash R(1)}(\mathbf{p}'; k') = \frac{+\backslash - \sum_{\mathbf{q}\in\mathcal{Q}} \mathbf{mv}(\mathbf{q}; k'\backslash k'-1) A_{(\mathbf{p}';k')}(\mathbf{q}; k'\backslash k'-1)}{\sum_{\mathbf{q}\in\mathcal{Q}} A_{(\mathbf{p}';k')}(\mathbf{q}; k'\backslash k'-1)}$$

3) Maximum overlap

$$\widehat{\mathbf{mv}}^{F\backslash R(1)}(\mathbf{p}'; k') = +\backslash - \mathbf{mv}(\mathbf{p}^*; k'\backslash k'-1)$$

and

$$\mathbf{p}^* = \arg\max_{\mathbf{q}\in\mathcal{Q}} A_{(\mathbf{p}';k')}(\mathbf{q}; k'\backslash k'-1)$$

4) Median

$$\widehat{\mathbf{mv}}^{F\backslash R(1)}(\mathbf{p}'; k') = +\backslash - \mathbf{mv}(\mathbf{p}^*; k'\backslash k'-1)$$

and

$$\mathbf{p}^* = \arg\min_{\mathbf{q}\in\mathcal{Q}} \left(\sum_{\mathbf{q}\in\mathcal{Q}} \|\mathbf{mv}(\mathbf{p}; k'\backslash k'-1) - \mathbf{mv}(\mathbf{q}; k'\backslash k'-1)\| \right)$$

where $\|\cdot\|$ denotes the Euclidean norm, the operators or expressions on the left and right sides of the backslash ($\backslash$) are for the fast-forward and fast-reverse video transcoding, respectively, and set $\mathcal{Q} = \mathcal{Q}(\mathbf{p}'; k'\backslash k'-1) = \{\mathbf{q} : A_{(\mathbf{p}';k')}(\mathbf{q}; k'\backslash k'-1) > 0\}$ contains the indices of the macroblocks in frame k' (for fast-forward transcoding) or frame $k'-1$ (for fast-reverse transcoding) involved in forming/referencing macroblock $\mathrm{MB}(\mathbf{p}'; k')$ in the pre-coded video. The performance of these fast motion vector re-estimation methods will be discussed in Section 4.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1217
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Intra-coding

Since the number of skipped frames increases with the playback speed-up factor, the temporal correlation between two successive frames in the transcoded video reduces as compared to that in the pre-coded video. In general, as the playback speed-up factor increases, the number of macroblocks which cannot find good correspondences in their reference frame also increases, especially for video with large and complex motion. In this case, instead of coding these macroblocks by a motion-compensated prediction scheme (i.e., inter-coding), they can be intra-coded to achieve a higher coding efficiency. However, when the speed-up factor is small, the video is mainly inter-coded and a fast motion vector re-estimation method can be used to obtain the motion vectors required. The algorithm which decides whether a frame is intra- or inter-coded can be summarized as follows. Let

$$SAD(\widetilde{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p};k)) = \sum_{\mathbf{n}\in\mathcal{N}} |I(\mathbf{p}+\mathbf{n};k) - I(\mathbf{p}+\widetilde{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p};k)+\mathbf{n};k-\gamma)|$$

$$M(\mathbf{p};k) = \frac{1}{|\mathcal{N}|}\sum_{\mathbf{n}\in\mathcal{N}} I(\mathbf{p}+\mathbf{n};k)$$

$$D(\mathbf{p};k) = \sum_{\mathbf{n}\in\mathcal{N}} |I(\mathbf{p}+\mathbf{n};k) - M(\mathbf{p};k)|$$

where $SAD(\widetilde{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p};k))$ designates the sum of absolute difference between macroblocks $MB(\mathbf{p};k)$ and $MB(\mathbf{p}+\widetilde{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p};k);k-\gamma)$, $\mathcal{N} = \{(i,j): i,j = 0,\ldots,15\}$, $|\mathcal{N}| = 256$, $M(\mathbf{p};k)$ the mean intensity value of macroblock $MB(\mathbf{p};k)$, and $I(\mathbf{p};k)$ the intensity value of pixel $\mathbf{p}$ in frame k of the pre-coded video. For a given frame k, intra-coding is preferred over inter-coding, if

$$\frac{1}{|\mathcal{P}|}\sum_{\mathbf{p}\in\mathcal{P}} \mathbf{1}\left(SAD(\widetilde{\mathbf{mv}}^{F\backslash R(\gamma)}(\mathbf{p};k)) - D(\mathbf{p};k) > C\right) > T$$

where $\mathbf{1}(\cdot)$ is an indicator function which has value 1 or 0 depending on whether the argument condition is true or not, $\mathcal{P}$ is the set of all macroblocks within the transcoded frame, $|\mathcal{P}|$ is the total number of the macroblocks, and C and T are two pre-determined thresholds. In our experiments, C is set to 500, which is the same as the threshold used for deciding the coding-mode in the H.263 standard, and T is empirically set to 0.2 based on the results from a set of test videos.

3. SUBJECTIVE QUALITY TEST

As most transcoding processes introduce coding distortion, some loss in video quality is unavoidable, especially when the target bit rate is low and a fast motion vector re-estimation method is employed. It is therefore necessary to examine the minimum quality degradation that is perceptible to human eyes when a video is played back at different fast speeds.

We conducted a series of subjective tests to assess the effect of fast video playback on human perceptual acuity. Three videos (Foreman, News, and Stefan) were used in the test. To obtain a fast playback video, the original video frames were temporally downsampled at the desired speed, and then compressed with a standard H.263 encoder at PSNRs ranging from 28 dB to 39 dB. The DSCQS (Double Stimulus Continuous Quality Scale) test method of ITU-R BT.500-8 [5] was adopted to conduct the subjective tests.

Five subjects participated in the test. All subjects had normal visual acuity. They were research students in engineering disciplines and half of them had participated in experiments based on visual stimuli. We conducted the experiments in a well-lighted laboratory. A computer system equipped with a 17-inch high-quality LCD display and a high-power graphics card was used to show the test videos at different fast playback speeds. The subjects sat approximately 20 inches from the display so that they could clearly see the video details. The video sequences were displayed in CIF format and placed at the center of a mid gray screen.

At the beginning of a test session, each subject was presented with a few trial videos of different qualities so that the internal sensitivity scale of the subject could be established for reliable judgement. In the test, a series of video pairs, one high quality benchmark video and one low quality transcoded video in each pair, were shown side-by-side to each subject at various fast playback speeds. The subject was then asked to decide which video appeared to have a lower quality. It should be mentioned that the videos were shown to the subject not in any particular order, and some tests were also randomly repeated, without the subject's awareness, in order to obtain consistent and reproducible results from each subject.

Table 1 summarizes the average of the minimum PSNR degradations that are perceptible to five subjects at various fast video playback speeds. A few observations from the results are in order: 1) The faster the playback speed, the larger the minimum PSNR degradation that can be perceived by the subjects; 2) When the speed-up factor grows beyond a certain value, the minimum perceptible PSNR degradation changes slowly; 3) Video with high or complex motion content can withstand a larger PSNR degradation before becoming perceptible to the subjects. For example, test video "Stefan" can withstand a larger PSNR degradation before it becomes perceptible when compared to test video "News".

The results suggest that a fast video transcoding method can be used to realize fast-forward or fast-reverse video playback at various fast speeds without apparent loss in quality, as long as the PSNR degradation incurred is smaller than that perceptible to the viewers.

Speed-up factor	1	2	3	4	5	6	7	8	9
Foreman (dB)	4.3	5.1	5.8	6.0	6.9	7.1	7.1	7.1	7.4
News (dB)	2.1	3.0	3.3	3.8	4.0	4.2	4.6	4.9	5.3
Stefan (dB)	9.3	9.3	9.6	10.5	11.3	12.8	13.2	13.8	13.9

Table 1. The average minimum PSNR degradations that are perceptible to the five subjects at different fast video playback speeds.

4. EXPERIMENTAL RESULTS

We report the experimental results in this section to show the performance of the proposed video transcoder. In our experiments, four test videos - Foreman, Stefan, Mobile & Calendar (M&C), and BBC - are pre-coded as H.263 bit streams with CIF format (352 × 288), 30 frames/second, and IPPP$\cdots$ GOP structure. Each GOP in the pre-coded videos has 60 frames. The motion vector of each 16 × 16 macroblock in the pre-coded video is obtained by using a full-search, block-matching motion estimation scheme with a 16×16 search window and full-pixel accuracy. The videos are then transcoded to various fast forward and reverse videos with speed-up factors ranging from 1 to 10, using the full-search motion estimation scheme and also the proposed fast motion vector re-estimation methods. The results obtained by the full-search motion estimation scheme are used as benchmark. The PSNR of each transcoded video is computed by comparing the transcoded video frames with the original, uncompressed video frames. Table 2 summarizes the bitrates of the pre-coded videos and the target transcoded videos.

Test video	Foreman	Stefan	M&C	BBC
Original bitrate (Kbps)	512	1500	2048	1500
Target bitrate (Kbps)	256	512	768	768

Table 2. Bit rate setting for test videos used in the experiments.

Figure 2 shows the PSNR results of test video "BBC", which is transcoded by different fast motion vector re-estimation methods for fast-reverse playback at 3 times the normal speed. The average PSNR values obtained by the "full-search", "maximum overlap", "area-weighted average", "median" and "in-place" methods are

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1218
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

about 27.95 dB, 26.99 dB, 26.74 dB, 25.53 dB and 26.18 dB, respectively. Considering the subjective test results reported in Section 3, the average 1 dB difference between the videos obtained by "full-search" and "maximum overlap" methods should not be visible in fast playback. Hence, instead of using "full-search" motion estimation, the "maximum overlap" method can be used to reduce the transcoding complexity.

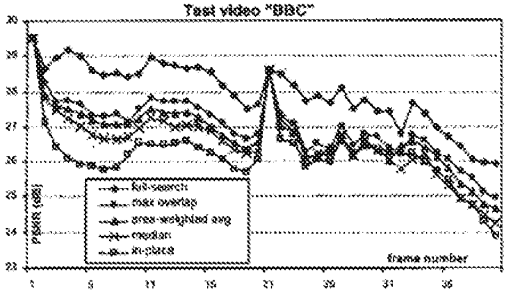
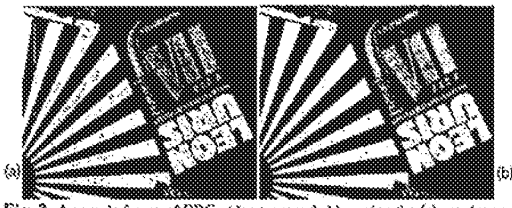


Fig. 2. PSNRs of the fast-reverse transcoded videos obtained by full-search and different fast motion vector re-estimation methods.



(a) (b)

Fig. 3. A sample frame of BBC video transcoded by using the (a) maximum overlap (27.75 dB) and (b) full-search (29.17 dB) methods.

Figure 3 shows a sample frame of test video "BBC" transcoded using the "full-search" and "maximum overlap" motion estimation methods, with PSNRs 27.75 dB and 29.17 dB, respectively. Although it is possible to tell the difference between these two still images at close inspection, it is rather difficult to do so when the videos are played back, resulting in a speed-up factor of 3. Table 3 shows the average PSNRs of the four test videos transcoded with a speed-up factor of 2 using the "full-search" and the proposed fast motion vector re-estimation methods. The results show that the "maximum overlap" performs consistently the best among all the fast methods examined. The average PSNR difference between the "full-search" and the "maximum overlap" methods is less than 1.3 dB. It is noteworthy that the simplest method, "in-place", does not necessary give the lowest PSNR. The reason is that most motion vectors in the pre-coded videos have small values and hence the corresponding macroblocks found by the "in-place" and "maximum overlap" methods are mostly the same. Hence, videos with stationary scene can be transcoded by "in-place" method for low computational complexity, while videos with high and complex motion by "maximum overlap" method for good quality.

Test video		Full-search	Max. overlap	Area-weighted avg	Median	In-place
Fast-forward	Foreman	31.51	30.85	29.75	29.63	30.62
	Stefan	27.75	26.56	26.18	26.32	26.44
	M&C	24.33	23.53	23.17	23.31	23.53
	BBC	27.33	26.86	25.83	25.47	25.33
Fast-reverse	Foreman	31.23	30.86	30.17	30.34	30.86
	Stefan	26.94	26.19	25.84	25.86	26.19
	M&C	24.25	23.04	22.81	22.66	23.03
	BBC	27.29	26.2	25.82	26.09	26.17

Table 3. Average PSNR (in dB) of the test videos transcoded by different fast motion vector re-estimation methods at a speed-up factor of 2.

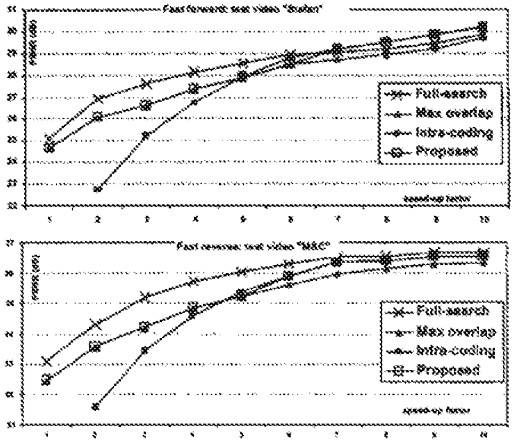


Fig. 4. The PSNRs of the fast-forward transcoded video "Stefan" and fast-reverse transcoded video "Mobile & Calendar" obtained by using full-search, maximum overlap, pure intra-coding, and the proposed adaptive transcoding method at different fast playback speeds.

Figure 4 shows the average PSNR results for fast-forward playback of test video "Stefan" and fast reverse playback of test video "Mobile & Calendar" with speed-up factors ranging up to 10. "Full-search", "maximum overlap", pure "intra-coding", and the proposed adaptive method (which combines intra-coding and inter-coding with "maximum overlap" fast motion vector re-estimation method) are used to transcode the test videos. It can be seen that "full-search" method performs the best at low speed-up factors, while pure "intra-coding" method performs better at high speed-up factors. The proposed adaptive approach automatically selects the inter-coding with "maximum overlap" fast motion estimation method at low speed-up factors and intra-coding method at high speed-up factors.

5. CONCLUDING REMARKS

We have proposed in this paper an efficient video transcoding approach for realizing fast forward/reverse video playback functions. In particular, several fast motion vector re-estimation methods for generating the new motion vectors required in the transcoded videos have been designed and compared. An adaptive video transcoding method is then proposed to combine the merits of intra-coding and inter-coding with a fast motion vector re-estimation scheme. We have also conducted subjective tests to assess the minimum quality degradation that is perceptible to human eyes when a video is played back at different fast speeds. Finally, experimental results are reported to show the effectiveness of the proposed method.

6. REFERENCES

[1] C.-W. Lin *et al.*, "MPEG video streaming with VCR-functionality," *IEEE Trans. on Circ. and Syst. for Video Technology*, Vol. 11, pp. 415-425, Mar. 2001.

[2] J.-N. Hwang *et al.*, "Dynamic Frame-skipping in Video Transcoding," *IEEE Workshop on Multimedia Signal Processing*, pp. 616-621, 1998.

[3] S. Wee, "Reversing motion vector fields," *International Conference on Image Processing*, pp. 209-212, Oct. 1998.

[4] B. A. Wandell, "Foundations of Vision," *Sinauer Associates, Inc.* 1995.

[5] ITU-R Recommendation BT.500-8, "Methodology for the subjective assessment of the quality of television pictures," Sept. 1998.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1219
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Archived content. No warranty is made as to technical accuracy. Content may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

Microsoft DirectX 8.1 (C++)

AVI RIFF File Reference

The Microsoft AVI file format is a RIFF file specification used with applications that capture, edit, and play back audio-video sequences. In general, AVI files contain multiple streams of different types of data. Most AVI sequences use both audio and video streams. A simple variation for an AVI sequence uses video data and does not require an audio stream.

Modifications to the original AVI file specification made in the OpenDML AVI File Format Extensions are not discussed in this section. For further information on these extensions, see version 1.02 of the *OpenDML AVI File Format Extensions* published by the OpenDML AVI M-JPEG File Format Subcommittee, February 28, 1996.

This section contains the following topics.

- AVI RIFF Form
- AVI Main Header
- AVI Stream Headers
- Stream Data (LIST 'movi' Chunk)

See Also

- **BITMAPINFOHEADER** Structure
- **WAVEFORMATEX** Structure

AVI RIFF Form

AVI files use the AVI RIFF form. The AVI RIFF form is identified by the FOURCC (four-character code) 'AVI '. All AVI files include two mandatory LIST chunks. These chunks define the format of the stream and stream data. AVI files might also include an index chunk. This optional chunk specifies the location of data chunks within the file. An AVI file with these components has the following form:

```
RIFF ('AVI '
      LIST ('hdrl'
               .
               .
               .
            )
      LIST ('movi'
               .
               .
               .
            )
      ['idx1'<AVI Index>]
     )
```

The LIST chunks and the index chunk are subchunks of the RIFF 'AVI ' chunk. The 'AVI ' chunk identifies the file as an AVI RIFF file. The LIST 'hdrl' chunk defines the format of the data and is the first required LIST chunk. The LIST 'movi' chunk contains the data for the AVI sequence and is the second required LIST chunk. The 'idx1' chunk is the index chunk. AVI files must keep these three components in the proper sequence.

The LIST 'hdrl' and LIST 'movi' chunks use subchunks for their data. The following example shows the AVI RIFF form expanded with the chunks needed to complete the LIST 'hdrl' and LIST 'movi' chunks:

```
RIFF ('AVI '
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1220
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
LIST ('hdrl'
      'avih'(<Main AVI Header>)
      LIST ('strl'
            'strh'(<Stream header>)
            'strf'(<Stream format>)
            'strd'(<additional header data>)
            'strn'(<Stream name>)
            ...
           )
       .
       .
       .
     )

LIST ('movi'
      (SubChunk | LIST ('rec '
                        SubChunk1
                        SubChunk2
                          .
                          .
                          .
                       )
         .
         .
         .
      )
      .
      .
      .
     )

['idx1'<AVI Index>]
)
```

AVI Main Header

This and following sections describe the chunks contained in the LIST 'hdrl' and LIST 'movi' chunks. The 'idx1' chunk is not described in this document. For more information on the 'idx1' chunk and indexes in AVI files, see version 1.02 of the *OpenDML AVI File Format Extensions* published by the OpenDML AVI M-JPEG File Format Subcommittee, February 28, 1996.

The file begins with the main header. In the AVI file, this header is identified by the 'avih' FOURCC (four-character code). The header contains global information for the entire AVI file, such as the number of streams within the file and the width and height of the AVI sequence. The AVI main header structure is defined as follows:

```
typedef struct {
    DWORD dwMicroSecPerFrame;
    DWORD dwMaxBytesPerSec;
    DWORD dwReserved1;
    DWORD dwFlags;
    DWORD dwTotalFrames;
    DWORD dwInitialFrames;
    DWORD dwStreams;
    DWORD dwSuggestedBufferSize;
    DWORD dwWidth;
    DWORD dwHeight;
    DWORD dwReserved[4];
} MainAVIHeader;
```

dwMicroSecPerFrame
Specifies the number of microseconds between frames. This value indicates the overall timing for the file.

dwMaxBytesPerSec
Specifies the approximate maximum data rate of the file. This value indicates the number of bytes per second the system must handle to present an AVI sequence as specified by the other parameters contained in the main header and stream header chunks.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1221
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

dwReserved1
Reserved. Set this to zero.

dwFlags
Contains any flags for the file. The following flags are defined.

Value	Description
AVIF_HASINDEX	Indicates the AVI file has an 'idx1' chunk containing an index at the end of the file. For good performance, all AVI files should contain an index.
AVIF_MUSTUSEINDEX	Indicates that the index, rather than the physical ordering of the chunks in the file, should be used to determine the order of presentation of the data. For example, you could use this to create a list of frames for editing.
AVIF_ISINTERLEAVED	Indicates the AVI file is interleaved.
AVIF_WASCAPTUREFILE	Indicates the AVI file is a specially allocated file used for capturing real-time video. Applications should warn the user before writing over a file with this flag set because the user probably defragmented this file.
AVIF_COPYRIGHTED	Indicates the AVI file contains copyrighted data and software. When this flag is used, software should not permit the data to be duplicated.

dwTotalFrames
Specifies the total number of frames of data in the file.

dwInitialFrames
Specifies the initial frame for interleaved files. Noninterleaved files should specify zero. If you are creating interleaved files, specify the number of frames in the file prior to the initial frame of the AVI sequence in this member. For more information about the contents of this member, see "Special Information for Interleaved Files" in the Video for Windows Programmer's Guide.

dwStreams
Specifies the number of streams in the file. For example, a file with audio and video has two streams.

dwSuggestedBufferSize
Specifies the suggested buffer size for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. If set to zero, or if it is too small, the playback software will have to reallocate memory during playback, which will reduce performance. For an interleaved file, the buffer size should be large enough to read an entire record, and not just a chunk.

dwWidth
Specifies the width of the AVI file in pixels.

dwHeight
Specifies the height of the AVI file in pixels.

dwReserved
Reserved. Set this array to zero.

AVI Stream Headers

One or more 'strl' chunks follow the main header. (A 'strl' chunk is required for each data stream.) These chunks

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1222
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

contain information about the streams in the file. Each 'strl' chunk must contain a stream header and stream format chunk. Stream header chunks are identified by the FOURCC (four-character code) 'strh' and the stream format chunks are identified by the FOURCC 'strf'. In addition to the stream header and stream format chunks, the 'strl' chunk might also contain a stream-header data chunk and a stream name chunk. Stream-header data chunks are identified by the FOURCC 'strd'. Stream name chunks are identified by the FOURCC 'strn'.

The stream header structure contains header information for a single stream of a file.

```
typedef struct {
    FOURCC fccType;
    FOURCC fccHandler;
    DWORD  dwFlags;
    DWORD  dwPriority;
    DWORD  dwInitialFrames;
    DWORD  dwScale;
    DWORD  dwRate;
    DWORD  dwStart;
    DWORD  dwLength;
    DWORD  dwSuggestedBufferSize;
    DWORD  dwQuality;
    DWORD  dwSampleSize;
    RECT   rcFrame;
} AVIStreamHeader;
```

The stream header specifies the type of data the stream contains, such as audio or video, by means of a FOURCC.

fccType
Contains a FOURCC that specifies the type of the data contained in the stream. The following standard AVI values for video and audio are defined.

'vids'	Indicates the stream contains video data. The stream format chunk contains a **BITMAPINFO** structure that can include palette information.
'auds'	Indicates the stream contains audio data. The stream format chunk contains a **WAVEFORMATEX** structure.
'txts'	Indicates the stream contains text data.

fccHandler
Optionally, contains a FOURCC that identifies a specific data handler. The data handler is the preferred handler for the stream. For audio and video streams, this specifies the installable compressor or decompressor.

dwFlags
Contains any flags for the data stream. The bits in the high-order word of these flags are specific to the type of data contained in the stream. The following standard flags are defined.

Value	Description
AVISF_DISABLED	Indicates this stream should not be enabled by default.
AVISF_VIDEO_PALCHANGES	Indicates this video stream contains palette changes. This flag warns the playback software that it will need to animate the palette.

dwPriority
Specifies priority of a stream type. For example, in a file with multiple audio streams, the one with the highest priority might be the default stream.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1223
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

dwInitialFrames
Specifies how far audio data is skewed ahead of the video frames in interleaved files. Typically, this is about 0.75 seconds. If you are creating interleaved files, specify the number of frames in the file prior to the initial frame of the AVI sequence in this member. For more information about the contents of this member, see "Special Information for Interleaved Files" in the Video for Windows Programmer's Guide.

dwScale
Used with **dwRate** to specify the time scale that this stream will use. Dividing **dwRate** by **dwScale** gives the number of samples per second. For video streams, this rate should be the frame rate. For audio streams, this rate should correspond to the time needed for **nBlockAlign** bytes of audio, which for PCM audio simply reduces to the sample rate.

dwRate
See **dwScale**.

dwStart
Specifies the starting time of the AVI file. The units are defined by the **dwRate** and **dwScale** members in the main file header. Usually, this is zero, but it can specify a delay time for a stream that does not start concurrently with the file.

dwLength
Specifies the length of this stream. The units are defined by the **dwRate** and **dwScale** members of the stream's header.

dwSuggestedBufferSize
Specifies how large a buffer should be used to read this stream. Typically, this contains a value corresponding to the largest chunk present in the stream. Using the correct buffer size makes playback more efficient. Use zero if you do not know the correct buffer size.

dwQuality
Specifies an indicator of the quality of the data in the stream. Quality is represented as a number between 0 and 10,000. For compressed data, this typically represents the value of the quality parameter passed to the compression software. If set to -1, drivers use the default quality value.

dwSampleSize
Specifies the size of a single sample of data. This is set to zero if the samples can vary in size. If this number is nonzero, then multiple samples of data can be grouped into a single chunk within the file. If it is zero, each sample of data (such as a video frame) must be in a separate chunk. For video streams, this number is typically zero, although it can be nonzero if all video frames are the same size. For audio streams, this number should be the same as the **nBlockAlign** member of the **WAVEFORMATEX** structure describing the audio.

rcFrame
Specifies the destination rectangle for a text or video stream within the movie rectangle specified by the **dwWidth** and **dwHeight** members of the AVI main header structure. The **rcFrame** member is typically used in support of multiple video streams. Set this rectangle to the coordinates corresponding to the movie rectangle to update the whole movie rectangle. Units for this member are pixels. The upper-left corner of the destination rectangle is relative to the upper-left corner of the movie rectangle.

The last eight members describe the playback characteristics of the stream. These factors include the playback rate (**dwScale** and **dwRate**), the starting time of the sequence (**dwStart**), the length of the sequence (**dwLength**), the size of the playback buffer (**dwSuggestedBuffer**), an indicator of the data quality (**dwQuality**), and the sample size (**dwSampleSize**).

Some of the members in the stream header structure are also present in the main header structure. The data in the main header applies to the whole file, while the data in the stream header structure applies only to a stream.

A stream format ('strf') chunk must follow a stream header ('strh') chunk. The stream format chunk describes the format of the data in the stream. For video streams, the information in this chunk is a **BITMAPINFO** structure

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1224
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

(including palette information if appropriate). For audio streams, the information in this chunk is a **WAVEFORMATEX** structure.

The 'strl' chunk might also contain an additional stream-header data ('strd') chunk. If used, this chunk follows the stream format chunk. The format and content of this chunk is defined by installable compression or decompression drivers. Typically, drivers use this information for configuration. Applications that read and write RIFF files do not need to decode this information. They transfer this data to and from a driver as a memory-block.

The optional 'strn' stream name chunk provides a zero-terminated text string describing the stream. (The AVI file functions can use this chunk to let applications identify the streams they want to access by their names.)

An AVI player associates the stream headers in the LIST 'hdrl' chunk with the stream data in the LIST 'movi' chunk by using the order of the 'strl' chunks. The first 'strl' chunk applies to stream 0, the second applies to stream 1, and so forth.

For example, if the first 'strl' chunk describes the wave audio data, the wave audio data is contained in stream 0. Similarly, if the second 'strl' chunk describes video data, then the video data is contained in stream 1.

Stream Data (LIST 'movi' Chunk)

Following the header information is a LIST 'movi' chunk that contains chunks of the actual data in the streams - that is, the pictures and sounds themselves. The data chunks can reside directly in the LIST 'movi' chunk or they might be grouped into 'rec' chunks. The 'rec' grouping implies that the grouped chunks should be read from disk all at once. This is used only for files specifically interleaved to play from CD-ROM.

Like any RIFF chunk, the data chunks contain a FOURCC (four-character code) to identify the chunk type. A FOURCC is a 32-bit quantity represented as a sequence of one to four ASCII alphanumeric characters, padded on the right with blank characters. The FOURCC that identifies each chunk consists of the stream number and a two-character code that defines the type of information encapsulated in the chunk. For example, a waveform chunk is identified by a two-character code of 'wb'. If a waveform chunk corresponded to the second LIST 'hdrl' stream description, it would have a FOURCC of '01wb'.

> **Note** While two-character codes are a convenient way to describe a stream, do not expect them to be recognized by other applications. Use FOURCCs when creating a stream or transferring the information to other applications.

Because all the format information is in the header, the audio data contained in these data chunks does not contain any information about its format. An audio data chunk has the following format (the ## in the format represents the stream identifier):

```
WAVE Bytes '##wb'
     BYTE  abBytes[];
```

Video data can be compressed or uncompressed DIBs. An uncompressed DIB has BI_RGB specified for the **biCompression** member in its associated **BITMAPINFO** structure. A compressed DIB has a value other than BI_RGB specified in the **biCompression** member. For more information about compression formats, see the description of the **BITMAPINFOHEADER** data structure in the Microsoft Windows Programmer's Reference.

A data chunk for an uncompressed DIB contains RGB video data. These chunks are identified by a two-character code of 'db' (db is an abbreviation for DIB bits). Data chunks for a compressed DIB are identified by a two-character code of 'dc' (dc is an abbreviation for DIB compressed). Neither data chunk will contain any header information about the DIBs. The data chunk for an uncompressed DIB has the following form:

```
DIB  Bits  '##db'
     BYTE  abBits[];
```

The data chunk for a compressed DIB has the following form.

BEST AVAILABLE COPY

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1225
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
Compressed DIB  Bits  '##dc'
                BYTE  abBits[];
```

Video data chunks can also define new palette entries used to update the palette during an AVI sequence.

Text streams can use arbitrary two-character codes.

Manage Your Profile | Legal | Contact Us | MSDN Flash Newsletter

© 2006 Microsoft Corporation. All rights reserved. Terms of Use | Trademarks | Privacy Statement

Microsoft

BEST AVAILABLE COPY

TCOM 570 1999-9

ENTROPY AND SOURCE CODING (COMPRESSION)

1. Introduction

Consider a message sequence of binary digits, or more generally a sequence of message symbols from a finite-size alphabet (e.g. English text), which is to be transmitted (or stored). The general sequence of symbols may be converted into a bit sequence by a simple symbol-to-character mapping such as provided by the ASCII code. Often we can find a ***more efficient representation*** of the sequence, which requires fewer bits to specify it than contained in the original sequence. This is because there is often much ***redundancy*** in message sequences in typical situations. We can seek ways of ***compressing*** the message into a shorter sequence ***without losing any information,*** that is in a reversible or ***loss-less way,*** so that the original sequence can be reconstructed upon receipt of its compressed version.

In English text, individual letters may be mis-typed or simply dropped at a fairly high rate and the text usually remains quite meaningful. This is because there is a lot of redundant information in much of English text. In fact most of the vowels can be dropped and it is still possible to make out many of the original words within the context of the message.

As a simple example of a sequence of symbols that can be compressed without any loss of information, the text string *"aaa...a"* in which there are 1000 *a*'s may be equivalently stated as "sequence of one thousand letter *a*". This requires 33 characters (including spaces) as opposed to 1000. This type of compression can be made even more efficient, and is an example of ***run-length coding*** in which repeated characters are replaced by an indication of the character and a repetition count. The use of the scientific notation 10^6 is another simple instance of this. In character oriented transmission, a string of identical characters occurring in the message can be replaced by an indication of the character value and a second character containing its repeat count (binary encoded integer), preceded by a reserved control character to indicate that character suppression is being used at that point in the message. This is a version of run-length coding. It requires three bytes to encode a string of repeated characters (and should be used for a string of three or more identical characters!). The old KERMIT personal computer file transfer protocol uses this scheme for compression.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1227
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. Lossy Compression

It is generally possible to achieve even more impressive compression if we are willing to lose some information permanently, that is if the compression is ***non-reversible***. Many speech and video compression schemes are ***"lossy"*** or non-reversible and the compressed information does not have the fidelity of the original. Nonetheless, some loss of information may be quite acceptable if the corresponding compression factor is very significant.

For example, instead of using 8-bit PCM for **telephone speech digitization** an alternative standard specifies differential PCM or **DPCM** that results in half the bit rate [32 Kbps] or less, instead of the 64 Kbps PCM rate, with an almost imperceptible loss of speech quality. In DPCM we transmit a quantized version of the ***difference*** between the current speech sample and the immediately previous reconstructed speech sample. The scheme starts with the first sample being quantized with the full 8-bit resolution of standard PCM. The first quantized sample is then subtracted from the next incoming sample and the difference is quantized with a 4-bit resolution. Because adjacent speech samples are correlated with each other, the differences tend to have a dynamic range that is smaller than the range of any one sample (differences tend to be smaller), and therefore a 4-bit quantization gives adequate representation of the differences. To reconstruct, we start with the first sample and then add to it the difference to get the second sample reconstruction, etc. Only the differences (after the first sample) are transmitted, thereby achieving the compression.

A similar principle can be used to transmit **video sequences.** A video source will typically produces 30 frames (still images) per second. Each frame is sampled to get a digital representation of the frame. A sample is called a picture element (pixel), and there could easily be 800x600 pixels per frame. Each pixel of a color image generally requires 8 bits for each of the three colors (R, G, B). This works out to a very high bit rate of 30x800x600x24 =345.6 Mbps! Even for grayscale (intensity only, no color) video we need more than 100 Mbps for transmission. Differential encoding can be very useful here also. Successive frames of the video are often very similar to each other, and pixel differences for pixels in corresponding positions in two frames can be encoded instead of pixel values. Furthermore, the effect of motion is generally to displace a block of pixels in one frame to a different location in the next frame. Video coders can make use of this by taking blocks (say of size 16x16) of each frame and comparing them to blocks in the previous frame to see if and where there is a close match. Upon finding a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1228
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

good match the displacement between the matching blocks is encoded, together with the differences between corresponding pixels in the current and matching previous block. This type of scheme (including other refinements not discussed here) can dramatically reduce the bit rate with small loss of quality. For example, a compression factor of 10 may be achieved easily with very good picture quality.

Our focus here will be on lossless schemes for data compression

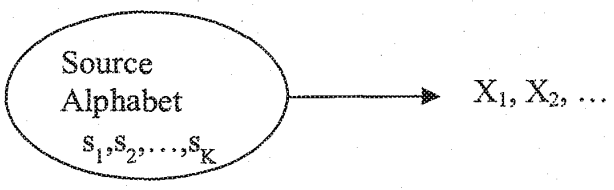
3. Information and Entropy

Important ideas in this section:

- *Quantitative definition of information*
- *entropy and its computation*
- *entropy as a bound for source compression*
- *the notion of a variable length code for compression, prefix condition codes, Huffman coding*
- *the possibility of significant compression for a source with non-equal symbol probabilities*

Consider a source producing a sequence of symbols $\{X_i , i=1, 2, ... \}$ from its alphabet set. In general, the source may have an alphabet of K letters $\{s_1, s_2, ..., s_K\}$. It emits some random sequence of letters from this alphabet as far as the destination is concerned. The output symbol X_i at time i is one of the letters in the source alphabet. A binary source has the symbol alphabet {0, 1}.

A ***discrete memoryless source*** (DMS) is a source with a finite alphabet for which the individual source output symbols X_i are ***statistically independent*** of each other. This is a simple model that is easy to work with.



Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1229
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- A fundamental question may be asked -- how much **"information"** is the source putting out each time it produces a symbol?

To see that this is not a vacuous question, consider the binary source and suppose it produces

letter s_1="0" with probability p

and

letter s_2="1" with probability 1-p.

Consider the special case in which p is very close to 1, say p=0.99. This means that with high probability the source just produces "0". If the receiver knew this probability distribution of letters, the source would be producing very little useful information as far as the receiver is concerned. This is because even if the source symbols were never transmitted to the receiver, the receiver would be able to reconstruct the source symbols to a high degree of accuracy by simply always assuming that the source symbol was "0". If the source was picking letters with some more uniform probability distribution (for example, equal probability for each letter in the source alphabet), the receiver would be getting more "information" each time a symbol got transmitted to it. This is because without the transmission the receiver would always be fairly uncertain as to the value of the symbol, and the transmission ***resolves this uncertainty*** by virtue of the information it has conveyed.

Information conveyed by a source may be quantified as ***a degree of uncertainty*** that one has about the outputs from a source before they are observed. Consider a particular letter "s" that gets produced with probability p. If p is small, the occurrence of "s" is very "informative". If p=1 the occurrence of "s" conveys no information since we already expect it to be observed.

We want therefore a measure of information conveyed that will vary in inverse relation to the probability p of occurrence. A measure that varies inversely as the probability p is $\log(1/p) = -\log(p)$. This quantity is always non-negative and decreases from a value of $+\infty$ at p=0, to 0 at p=1. We choose this definition for the information associated with the occurrence of a particular output "s" which has a probability p for the following reason. *We would like to have the property that a sequence of two symbols produced independently of each other from the same source should have an amount of information that is the sum of the individual information for each of the two symbols.* Since the probability of the sequence of two symbols is the product of the individual probabilities (product rule for joint occurrence of independent events), and the log of a product is the sum of the logs, we have the desired property.

- **The information associated with occurrence of a symbol "s" with *a priori* probability "p" of occurring is $-\log_2(p)$**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1230
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This leads to the following result for the **entropy, or average amount of information per symbol, H,** for a source of alphabet size N for which the individual letters are produced with probabilities p_n, n=1, 2, ..., K :

$$\mathbf{H} = -\sum_{n=1}^{K} p_n \log_2(p_n)$$

The **entropy** is simply the **expected value of the information** $-\log_2(p)$ associated with each output from the source. Notice that we have chosen to use the base 2 in the logarithm. With this choice of the log base, the **units of H are "bits per symbol".** The term "bit" usually stands for *"bi*nary dig*it"*, that is a "1" or a "0". Here the unit of entropy is defined as the "bit" when we use log base 2 in the definition; this is a unit of "information". It turns out that there is a close relationship between the entropy of a source in bits per source symbol and the minimum number of binary digits we need to represent each source symbol.

Numerical Example:

(a) Consider a **binary source** with $p_1 = p = 1/8$ and $p_2 = 1-p = 7/8$. Then entropy H for this source is, using the fact that $\log_2(a/b)=\log_2(a) - \log_2(b)$ and $\log_2 8 = 3$,

$H = -(1/8)\log_2(1/8) - (7/8)\log_2(7/8) = (3/8) + (21/8) - (7/8)\log_2(7)$
$= 0.54$ *bits/symbol.*

If the source had $p=1-p=1/2$ (equally likely letters) then

$H = -(1/2)\log_2(1/2) - (1/2)\log_2(1/2) =$ *1 bits/symbol*

(b) More generally, for a **source alphabet of size K and equally likely letters** we will have $\mathbf{H=\log_2 K}$ **bits/symbol.**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1231
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- We can also show that **H ≥ 0 always** and that the **maximum value for H** for a source with symbol alphabet of K letters is **$\log_2$K bits/symbol** ***(see accompanying book section for proof.)***

H is never negative (reasonable, given it is the expected value of –log(p), non-negative "information") and the largest average information is associated with a source with equally likely symbols. This is in agreement with our intuition, since for such a source it is most difficult to predict (we are most uncertain about) the value of the symbol it will emit.

4. Statistical Encoding for Compression

There is a fundamental set of results on compression of source sequences, and we will state a result in simple form here, without formal proof, that indicates the central importance of the concept of **entropy.**

Before we proceed note that if we have a source with symbols from an alphabet of size K and we do not utilize any statistical information about the source, then we would need a number of bits (binary digits) to represent it that is equal to the least integer upper bounding $\log_2$K. For example, suppose we have a source producing one of K=8 letters each time. We could use a 3-bit code, since 2^3=8 and $\log_2 8$ =3. If K=10 we would need a 4-bit code, since $\log_2 10$ =3.32 (We can produce 2^4=16 combinations of 4 binary digits, hence we can assign a unique 4-bit sequence to each of the 10 letters.)

A Source Compression Theorem:

"Consider a source producing **independent symbols from a finite alphabet** according to a specific probability distribution (e.g. a binary source producing a sequence of independent, identically distributed, binary digits). Let the **source entropy be H bits/symbol.** Then it is always possible to **compress** the source **without losing information** and get arbitrarily close to the **minimum possible average transmission rate of H binary digits (bits) per source symbol."**

This is a celebrated result of information/communication theory.
(see accompanying book section for details)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1232
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Justification of Connection between Entropy and Compression for DMS:

Consider a *long* sequence $\mathbf{X}=(X_1, X_2, \ldots, X_N)$ of symbols from a DMS. Each X_i is one of the K letters of the source alphabet. *With high likelihood* this sequence will contain close to Np_1 occurrences of the letter s_1, Np_2 occurrences of letter s_2, ... Np_K occurrences of letter s_K. (Note that N is large). With high likelihood sequences of length N produced by the source will have this composition; such sequences are the "typical sequences". The probability of producing any particular typical sequence **x** of length N is therefore

$$\mathrm{P}(\mathbf{X}=\mathbf{x}) \cong \prod_{n=1}^{K} p_n^{Np_n}$$

and

$$\log_2 \mathrm{P}(\mathbf{X}=\mathbf{x}) \cong \sum_{n=1}^{K} Np_n \log_2 p_n$$
$$= -NH$$

so that

$$\mathrm{P}(\mathbf{X}=\mathbf{x}) \cong 2^{-NH}$$

Thus *with high likelihood* the probability of a long sequence of length N is very close to 2^{-NH} This means that there are 2^{NH} such "typical" equiprobable (high likelihood) sequences; any other sequences are going to form a subset of probability close to 0. To specify one of these 2^{NH} typical sequences uniquely, we need NH bits. Thus on the average we need to use H bits per source symbol to specify with high probability all sequences that the source will produce.

Note that H binary digits per source symbol becomes the lowest achievable compression for such a DMS. Since H can be much smaller than $\log_2(K)$ for a source of alphabet size K, for example, this suggests the possibility of significant compression.

It turns out that if the source produces symbols that are statistically *dependent* than it is possible to achieve *even lower rates.* On the other hand, for independent symbols it is not possible to compress to less than H bits per source symbol on the average.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1233
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Example of Statistical Compression:

Consider a source producing independent symbols, from the alphabet of letters {A, B, C, D}, each time with the following probabilities:

A	with probability	$\frac{3}{4}$
B	with probability	$\frac{1}{8}$
C	with probability	$\frac{1}{16}$
D	with probability	$\frac{1}{16}$

The source entropy is

$$-(3/4)\log_2(3/4) - (1/8)\log_2(1/8) - (1/16)\log_2(1/16) - (1/16)\log_2(1/16)$$

$$= 1.186 \text{ bits/symbol}$$

For 4 letters we can use a simple two-bit code (00, 01, 10, and 11) and this would require 2 bits/symbol to represent the source outputs. But we can transmit the source using less than 2 bits/symbol on the average.

Notice that if the letter A is encoded using n_A bits, the letter B using n_B bits, etc., then the average number of bits per letter is $\frac{3}{4}\ n_A + \frac{1}{8}\ n_B + \frac{1}{16}\ n_C + \frac{1}{16}\ n_D$. The simple code above used 2 bits for each letter. Suppose we could find a code that uses ***fewer bits for the more probable letters,*** that is make n_A smaller than 2 and n_C and n_D larger than 2. Then we could expect a smaller average number of bits per symbol. In fact, consider the following code:

A	sent as	0	(1 bit)
B	sent as	10	(2 bits)
C	sent as	110	(3 bits)
D	sent as	111	(3 bits)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1234
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This is a **variable-length code,** assigning codewords of different lengths to different letters. Now the average length of the codeword, that is **the average number of bits per symbol, is**

$\frac{3}{4}\ 1 + \frac{1}{8}\ 2 + \frac{1}{16}\ 3 + \frac{1}{16}\ 3 = \mathbf{1.375}$ Compare this to the entropy H which is 1.186 bits/symbol. According to the theorem above we could do even better than this. We will see later how.

We also notice that it was the **non-uniform** nature of the source letter probability distribution that worked in our favor for this example. If the source letters had been equally likely, produced with probability $\frac{1}{4}$ each, then H would have been 2 bits/symbol and the simple 2-bit code would have been the best possible. Thus we have the potential of obtaining significant savings for sources producing letters that are not equally likely.

One might question how we would **decode** the bit stream produced by the variable length code that we defined in this example. Since the codewords do not have fixed sizes, how do we know the codeword boundaries in a transmitted bit stream? For the example above we see that ***no codeword was the prefix*** of any other codeword. Thus the codeword boundaries in the stream of transmitted bits can be obtained simply by looking for the first complete codeword that can be formed following start of transmission or after the end of the last complete codeword identified. The code satisfied the ***"prefix condition"*** that no codeword be the prefix of any other valid codeword.

See accompanying book section for details on prefix condition codes.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1235
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5. Huffman Coding

This is a **variable length coding** technique that creates a **prefix-condition** code given the non-uniform probability distribution of any finite alphabet. It provides a **systematic approach** to designing a variable length code which is **best** for a given finite-alphabet source (that is, a given probability distribution for its symbols). The example above is one instance of a code designed by this general technique. The sense in which this procedure results in a "best" code is that ***no other code can have a lower average number of bits per source symbol when individual source symbols are encoded one by one as they are produced.*** It is possible to do better by processing source symbols in ***blocks*** of 2 or more consecutive outputs from the source ***(compound symbols from an extended alphabet)***, as we shall see.

The mechanics of producing the Huffman code for a given probability distribution are quite simple. To illustrate, we will consider an example.

Example of Huffman Coding:

Suppose we have a source with alphabet size K=7. Let the source symbols be from the alphabet {A, B, C, D, E, F, G} with respective probabilities {0.3, 0.2, 0.15, 0.15, 0.08, 0.07, 0.05}. Trace the steps below in the figure on the next page.

Step 1

List the source letters in **decreasing** probability order. (The letters A through G for the example are already labeled in decreasing probability order).

Step 2

Combine the two least-probability letters into a single entity (call it "fg" in Stage 2 for the example), associating with it the **sum of the probabilities** of the individual letters in the combination. Assign to each of the letters that were combined one of the binary digits, "0" for upper symbol , "1" for the lower one (F →"0", G →"1" in the example).

Step 3

Re-order, if necessary, the resulting letters list in decreasing probability order, treating any combined letters from the previous list as a single letter. If the list now has only 2 letters, assign to each of them one of the binary digits (upper symbol →0, lower one →1) and go to Step 4. Otherwise go to Step 2.

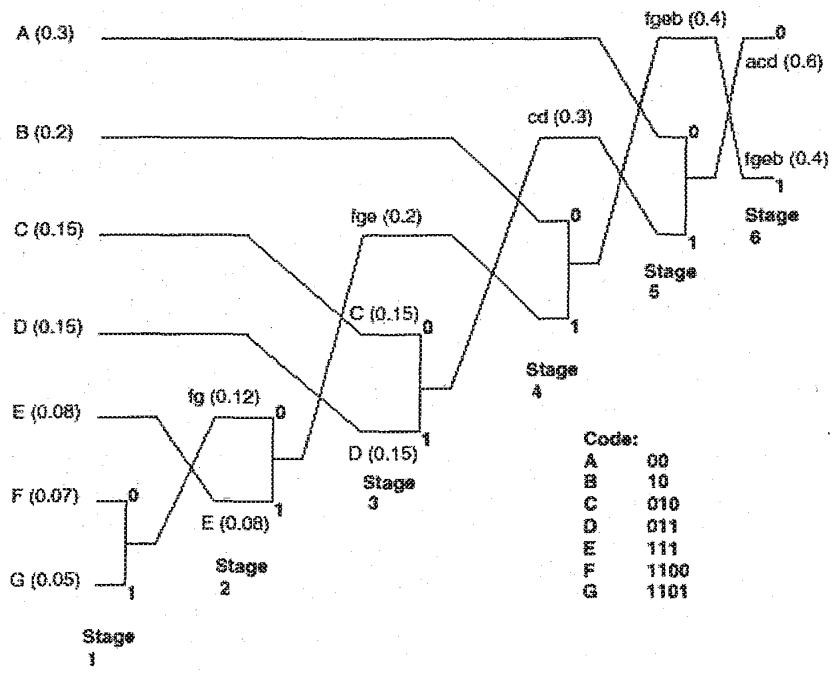
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1236
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Step 4

Read off the binary codewords for the source symbols, from right to left.

(Note: Assignment of code letters "0" and "1" to upper and lower letters in a pair may be done in any way as long as it is consistent throughout. If we assigned "1" for upper and "0" for lower letter, all the codewords would have "1" and "0" interchanged.)

It may now be verified that the code for the 4-symbol alphabet of the example in the previous section is a Huffman code. The Huffman code is also always a prefix-condition code, by construction.



Huffman Code Construction for Example Source

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1237
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1238
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Note on Re-Sorting at Each Stage

If at any stage in combining least-probability letters and re-sorting, the resulting combined probability has the same value as another letter above it (a "tie" occurs), there are two options. We may leave the combined-letter in the lowest position or we may place it in the highest position it could occupy. Each convention will result in a valid code and the two are equivalent in terms of average code word length; however the details of the codewords can be quite different.

Extension of Huffman Coding

Suppose in the example of the 4-letter source alphabet of **Section 4** the source produces statistically independent symbols according to the given probability distribution
$\{\frac{3}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}\}$. If we look at blocks of 2 symbols at a time, we get 16 combinations of letters {AA, AB, AC, AD, BA, ..., DD}, and the respective probabilities are the products of the individual probabilities of the letters making up the 2-letter blocks. We can apply the Huffman code construction to this extended source alphabet, and work out codewords for each of the 16 pairs of letters. The average codeword length for the 16-pair extended source may then be found, and dividing this result by 2 gives the average number of binary digits for each original source symbol. It turns out that this procedure gives better performance, i.e. a lower average number of bits per source symbol. The ultimate limit here is still the entropy rate of H bits per symbol.

(Carry out this procedure and verify this statement)

In general, we can get better performance by "blocking" symbols to define extended sources and using the Huffman code construction for the extended source. The complexity grows quickly, however. For example, for the English alphabet and some punctuation characters we might have 72 letters; for a 2-symbol extended source the alphabet size becomes 72^2 which is more than 4000. On the other hand, the extended source may give very significant further savings especially when the source produces correlated symbols to begin with.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1239
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Dynamic Huffman Coding

For Huffman coding, we need to know the probabilities for the individual letters (and we also need to know the joint probabilities for blocks of letters in extended Huffman coding). If the probability distributions are not known for a file of characters to be transmitted, they have to be estimated first and the code itself has to be included in the transmission of the coded file. In dynamic Huffman coding a particular probability (frequency of symbols) distribution is assumed at the transmitter and receiver and hence a Huffman code is available to start with. As source symbols come in to be coded the relative frequency of the different symbols is updated at both the transmitter and the receiver, and corresponding to this the code itself is updated. In this manner the code continuously adapts to the nature of the source distribution, which may change as time progresses and different files are being transferred.

Dynamic Huffman coding is the basis of data compression algorithms used in V series modems for transmission over the PSTN. In particular the MNP (Microcom Networking Protocol) Class 5 protocol commonly found in modems such as the V.32*bis* modem uses both **CRC error detection and dynamic Huffman coding for data compression.**
A modified form of coding together with run-length coding is used in compression for facsimile transmission.

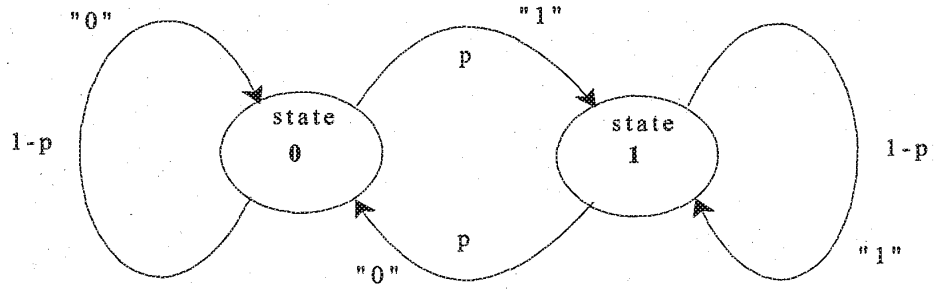
6. Other Aspects of Discrete Source Coding

Discrete Source with Memory

Consider a discrete source X with a finite alphabet, but now with some form of statistical dependence between successive symbols X_i from the source. If we have already observed the set of symbols up to time n–1, how much *additional* information on average can be obtained from X_n? For the memoryless source, conditioning on the past is irrelevant, and the amount of information is the entropy H. However, in the case of statistical dependence, conditioned on the past, the amount of information provided by X_n will be *less*; we are already able to predict to a certain extent what X_n will be, because of the statistical dependence. This would suggest that the definition of the entropy H for a discrete source with memory will give a result smaller than that for a corresponding source with the same distribution of letter probabilities, for which the symbols are independent of each other. Thus we should be able to compress such sources significantly.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1240
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Consider the following specific example of a simple binary *Markov* source. The source is always in one of two *states,* state "0" or state "1". If it is in state "0" at any time index, then with *probability (1–p)* it produces letter "0" and *stays in the same state,* and with *probability p* it produces letter "1" and *switches* to state "1". In state "1" it produces a "1" and stays in that state with probability (1– p), and it produces "0" and switches to state "0" with probability p. The state of the source at any time corresponds to its previous output value. Also, each time with probability (1–p) it will produce the same letter ("0" or "1") as it did the previous time. Notice that in the long run it will tend to produce equal numbers of "0" and "1", so that as far as source output at any one specific time index by itself is concerned, the probability of a "1" is the same as the probability of a "0" (equal to 1/2 for both). However, the source outputs are obviously heavily correlated if p is small or close to 1. For small p, the source tends to produce "runs" of letters.



If the source is in state "0", it produces the output letters with respective probabilities p and 1–p. With probability 1/2 it is in this state, and the entropy conditioned on this being the state is $-p\log_2 p - (1-p)\log_2 (1-p)$. Similarly for the other state, and the overall entropy is thus also $-p\log_2 p - (1-p)\log_2 (1-p)$. This function of p has a peak value of 1 at p=1/2; for other values of p the entropy is strictly lower than 1 bit/symbol. For small p we can use run-length coding to produce good compression factors. *(Note that the per-letter entropy of this source is 1 bit/symbol, according to the simple definition of H).*

Notice that for *any* source in general, its per-letter entropy can always be computed according to the earlier definition of entropy. Huffman coding may be used to achieve compression. For memoryless sources, the per-letter entropy is the same as entropy from the above point of view. For sources with memory, the actual entropy is lower than per-letter entropy. Better compression factors may be achieved; in Huffman coding with extended (block) alphabets, the probability distributions of letter-blocks become even more non-uniform and the entropy per block is less than the sum of the individual per-letter entropies.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1241
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Arithmetic Coding

We have seen that Huffman coding of a source (or an extended, blocked, source) can provide a compressed bit rate that can approach the entropy limit of H bits/symbol. For any fixed block size (one or more symbols per block) we may not be able to come as close as we would like to the entropy limit. This is because there is an inherent "granularity" in defining the Huffman code. If the probabilities of the alphabet of source are close to being of the form 2^{-m}, then we will get good performance very close to the entropy limit. In general this will not be true. In addition, in using the Huffman scheme to encode a source we first have to obtain the complete code before any source letters can be encoded. This may be rather difficult for a large-alphabet source.

Arithmetic coding is an interesting approach to coding that can avoid both of these problems. Below is an illustration of how arithmetic coding works.

In arithmetic coding we construct a *code interval* within the unit interval [0,1] that uniquely represents a block of letters from a source. We begin by assigning to each letter of the alphabet an interval of length equal to its probability. Let the letter s_n has probability p_n. Suppose we have K=4 and p_1=0.5, p_2=0.3, p_3=0.15, and p_4=0.05. To letter s_1 we assign the interval $[0,p_1)$=[0, 0.5), to s_2 the interval $[p_1, p_1+p_2)$=[0.5, 0.8), to s_3 the interval [0.8, 0.95) and to s_4 the interval [0.95, 1). These are known to both coder and decoder. Now to encode a block of letters $\{s_2, s_1, s_1, s_4, s_3\}$ say, we start with the interval [0.5, 0.8) for s_2, *within it* locate the interval for s_1 as [0.5, 0.65), within it s_1 again as [0.5, 0.575), within it s_4 as [0.57125, 0.575), and within it s_3 as [0.57425, 0.5748125). This final interval uniquely identifies the five-letter block. In fact *any* value within this interval uniquely identifies the five-letter block. In particular, we may pick the 11-bit fraction (implied binary point at left) representing 0.57470703125 inside this interval (this is a "tag" for this interval).

This is the general procedure for arithmetic coding, with a similar decoding operation. Arithmetic coding provides a good way to achieve efficient data compression codes; it is more practical and lends itself more easily to adaptive schemes for changing source statistics.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1242
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

7. The LZW Algorithm

The Lempel-Ziv-Welch algorithm is a scheme for lossless compression of character or byte sequences. It is a very popular technique for compression, and is used in compression of text files for storage in computer systems (e.g. the Unix COMPRESS and GZIP utilities) and also in transmission of data via modems.

The basic idea of the whole class of such algorithms, of which LZW is one particular version, is the following. In sending or storing a string of characters, character patterns that have occurred earlier on in the string need not be sent, only a code referring to the pattern needs to be used. For example, you could create a dictionary of patterns that you encounter as you see new characters of the string, and assign a codeword for each new pattern, or perhaps only to each frequently used pattern. As an example, we could decide that from this point on in this set of notes the word "compression" will be designated by a special codeword say !c, and the word "codeword" will be designated as ?c The procedure for building the code dictionary can be designed such that the receiver (decoder or decompressor) can also construct the same dictionary that the transmitter (encoder or compressor) is constructing as characters come in, so that it is able to recover the original string. This is best illustrated by an example of the operation of the LZW coder/decoder.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1243
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

LZW Coding

Assume an alphabet { _ a b o d } of size 5 from which text characters are picked. Suppose the following string of characters is to be transmitted (or stored):

d a b b a _ d a b b a _ d a b b a _ d o o _ d o o_ .....

The coder starts with an initial dictionary consisting of the 5 individual letters of the alphabet and an index assignment, as in the following table:

ENTRY	INDEX
_	1
a	2
b	3
o	4
d	5

The first symbol that the encoder gets is the letter **"d"**. This is in the table, so it looks at the next symbol, which is letter **"a"**. The sequence **"da"** is not in the table, so the code for the first symbol **"d"** is sent (Index **5** is sent) and the sequence **"da"** is added to the table with index = **6**. The symbol waiting to be sent is **"a"**

The coder then looks at the next (third) symbol, which is a **"b"**. The sequence **"ab"** is not in the table, so **"a"** is sent as index **2** and **"ab"** is added to the table as index **7**. The symbol waiting to be sent is **"b"**

The next symbol is **"b"**. Sequence **"bb"** is not in the table. Letter **"b"** is sent as index **3**, and **"bb"** is added to the table as index **8**. The symbol waiting to be sent is **"b"**

The next symbol is **"a"**. Sequence **"ba"** is not in the table. Letter **"b"** is sent as index **3**, and **"ba"** is added to the table as index **9**. The symbol waiting to be sent is **"a"**

The next symbol is **"_"**. Sequence **"a_"** is not in the table. Letter **"a"** is sent as index **2**, and **"a_"** is added to the table as index **10**. The symbol waiting to be sent is **"_"**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1244
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The next symbol is **"d"**. Sequence **"_d"** is not in the table. Letter **"_"** is sent as index **1**, and **"_d"** is added to the table as index **11**. The symbol waiting to be sent is **"d"**

The next symbol is **"a"**. Sequence **"da"** is in the table. The next symbol is **"b"**. Sequence **"dab"** is not in the table. Sequence **"da"** is sent as index **6**, and **"dab"** is added to the table as index **12**. The symbol waiting to be sent is **"b"**

The next symbol is **"b"**. Sequence **"bb"** is in the table. The next symbol is **"a"**. Sequence **"bba"** is not in the table. **"bb"** is sent as index **8**, and **"bba"** is added to the table as index **13**. The symbol waiting to be sent is **"a"**

The next symbol is **"_"**. Sequence **"a_"** is in the table. The next symbol is **"d"**. The sequence **"a_d"** is not in the table. Sequence **"a_"** is sent as index **10**, and **"a_d"** is added to the table as index **14**. The symbol waiting to be sent is **"d"**

The transmitted indices so far are: 5 2 3 3 2 1 6 8 10

The dictionary table at this point looks like this:

ENTRY	INDEX
_	1
a	2
b	3
o	4
d	5
da	6
ab	7
bb	8
ba	9
a_	10
_d	11
dab	12
bba	13
a_d	14

Continuing this way, the transmitter now looks at the next symbols **"abb"** beyond the **"d"** waiting to be sent. It sends out **"dab"** as index **12**, and adds **"dabb"** to the table as entry **15**. The symbol **"b"** is now waiting. The symbols **"a_"** come in, and **"ba"** is sent as **9**, with **"ba_"** added to the table as **16**. The symbol **"_"** is now waiting.

The next symbols **"do"** cause the transmitter to send **"_d"** as **11** and add **"_do"** as index **17** in the table. The next incoming symbol **"o"** causes **"oo"** to be added to the table as **18** and index **4** to be sent out. Symbol **"o"** is waiting.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1245
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

In response to the next symbols "_" and **"doo"** the transmitter adds **"o_"** and **"_doo"** to the table, and sends index codes **4** and **17.** The rest of the procedure depends on the other incoming symbols.

The added table items at this point are:

ENTRY	INDEX
dabb	15
ba_	16
_do	17
oo	18

The transmitted indices to this point are:

5 2 3 3 2 1 6 8 10 12 9 11 4 4 17

Notice, as the table builds up longer sequences appear in the dictionary indicating that compression performance will get better as the coder builds up more information about the nature of the source, i.e. what type of strings it produces.

In practice of course the index values are transmitted as binary words. For example, fixed length 16-bit words may be used to represent the index values as binary coded integers. (With 16 bits, a table of size approx. 64,000 can be built)

LZW Decoding

Consider the decoder operation when the above index string is received. The decoder starts off with the same 5-index single-letter table as does the coder. The initial indices 5 2 3 3 2 1 allow the decoder to obtain the first few symbols (dabba_), and at the same time as the decoder decodes these symbols it expands its table exactly as the coder builds up its table. Thus when the next index 6 comes in the decoder knows that this represents the string "da"

Notice that the coder adds to the dictionary when it sees a symbol that would create a new sequence in the dictionary.

(Special case handling: Can the decoder receive an index code for a sequence not yet in its table? What can happen is that the decoder gets an index for a sequence it has not yet completed for addition to the table. This

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1246
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

is because the coder looks one symbol ahead for the current transmission and update to its table, but the decoder does not have current access to the next symbol. Suppose the transmitted sequence was simply "_a_a_a_a_a_a....." Then the coder output will be 1 2 6 8 7 10 ... and the coder table would build up as:

ENTRY	INDEX
_	1
a	2
b	3
o	4
d	5
_a	6
a_	7
_a_	8
_a_a	9
a_a	10

At the decoder, the received 1 2 6 will result in the table being built up to index 7, with a partial sequence "_a" waiting to be completed as an entry for index 8. But at this point the index 8 is received, and we do not yet know what it represents! In this case the procedure is simply to use the partial sequence for 8, which is "_a", and note that the waiting "_a" makes a new entry with the first "_" of the partial result for 8. Thus 8 is decoded as "_a_". At this point the "_a_" remains as the beginning of the next index entry (9).

When 7 comes in next, it can be decoded and the sequence "_a_a" is entry 9. At this point "a_" is the sequence waiting to be completed as 10, when 10 comes in. This makes entry 10 "a_a", etc.

One way to avoid this special case is to not allow use of the very last table index, but require transmission of codewords for the constituents of the last entry. This avoids the use at the coder of an index that was added in its immediately previous step, since the decoder does not yet have access to the symbol used in its determination.)

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1247
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Unix COMPRESS:

This uses an initial dictionary table of size 512 (9 bit codes). When this is filled, the next 512 dictionary indices are encoded as 10 bit words, etc. This way the shorter words are used for the initial shorter dictionary sequences. A maximum of 9 to 16 bits are specified by the user for the codeword size (default 16). When the maximum table size has been reached, the table remains fixed and the scheme becomes a static table-look-up coding scheme. The algorithm now monitors the actual compression ratio attained, and if it falls below a limit the table building is re-started. This allows the scheme to track the nature of the text in the file being compressed.

Modem Compression

The V.42bis modem compression standard uses a form of LZW coding. Some of the particular characteristics include:

- Periodic testing of compression factor; algorithm can switch between compression and transparent mode, depending on compression performance.
- Variable dictionary size, initial size negotiated at start-up. Recommended 2048, minimum 512.
- Special codewords for "enter transparent mode" and "increment dictionary index size by one bit"
- When dictionary size reaches maximum, strings not encountered recently are pruned.
- Maximum string length is 6 to 250, default 6 (Maximum length is imposed to control errors, since for long strings an error in its codeword causes loss of many characters).
- Decoding special case is avoided by not allowing use of latest entry in dictionary at any step.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1248
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Image Quality Assessment: From Error Visibility to Structural Similarity

Zhou Wang, *Member, IEEE*, Alan Conrad Bovik, *Fellow, IEEE*, Hamid Rahim Sheikh, *Student Member, IEEE*, and Eero P. Simoncelli, *Senior Member, IEEE*

***Abstract*—Objective methods for assessing perceptual image quality traditionally attempted to quantify the visibility of errors (differences) between a distorted image and a reference image using a variety of known properties of the human visual system. Under the assumption that human visual perception is highly adapted for extracting structural information from a scene, we introduce an alternative complementary framework for quality assessment based on the degradation of structural information. As a specific example of this concept, we develop a Structural Similarity Index and demonstrate its promise through a set of intuitive examples, as well as comparison to both subjective ratings and state-of-the-art objective methods on a database of images compressed with JPEG and JPEG2000.[1]**

***Index Terms*—Error sensitivity, human visual system (HVS), image coding, image quality assessment, JPEG, JPEG2000, perceptual quality, structural information, structural similarity (SSIM).**

I. Introduction

DIGITAL images are subject to a wide variety of distortions during acquisition, processing, compression, storage, transmission and reproduction, any of which may result in a degradation of visual quality. For applications in which images are ultimately to be viewed by human beings, the only "correct" method of quantifying visual image quality is through subjective evaluation. In practice, however, subjective evaluation is usually too inconvenient, time-consuming and expensive. The goal of research in *objective* image quality assessment is to develop quantitative measures that can automatically predict perceived image quality.

An objective image quality metric can play a variety of roles in image processing applications. First, it can be used to dynamically *monitor* and adjust image quality. For example, a network digital video server can examine the quality of video being transmitted in order to control and allocate streaming resources. Second, it can be used to *optimize* algorithms and parameter settings of image processing systems. For instance, in a visual communication system, a quality metric can assist in the optimal design of prefiltering and bit assignment algorithms at the encoder and of optimal reconstruction, error concealment, and postfiltering algorithms at the decoder. Third, it can be used to *benchmark* image processing systems and algorithms.

Objective image quality metrics can be classified according to the availability of an original (distortion-free) image, with which the distorted image is to be compared. Most existing approaches are known as *full-reference*, meaning that a complete reference image is assumed to be known. In many practical applications, however, the reference image is not available, and a *no-reference* or "blind" quality assessment approach is desirable. In a third type of method, the reference image is only partially available, in the form of a set of extracted features made available as side information to help evaluate the quality of the distorted image. This is referred to as *reduced-reference* quality assessment. This paper focuses on full-reference image quality assessment.

The simplest and most widely used full-reference quality metric is the mean squared error (MSE), computed by averaging the squared intensity differences of distorted and reference image pixels, along with the related quantity of peak signal-to-noise ratio (PSNR). These are appealing because they are simple to calculate, have clear physical meanings, and are mathematically convenient in the context of optimization. But they are not very well matched to perceived visual quality (e.g., [1]–[9]). In the last three decades, a great deal of effort has gone into the development of quality assessment methods that take advantage of known characteristics of the human visual system (HVS). The majority of the proposed perceptual quality assessment models have followed a strategy of modifying the MSE measure so that errors are penalized in accordance with their visibility. Section II summarizes this type of error-sensitivity approach and discusses its difficulties and limitations. In Section III, we describe a new paradigm for quality assessment, based on the hypothesis that the HVS is highly adapted for extracting structural information. As a specific example, we develop a measure of structural similarity (SSIM) that compares local patterns of pixel intensities that have been normalized for luminance and contrast. In Section IV, we compare the test results of different quality assessment models against a large set of subjective ratings gathered for a database of 344 images compressed with JPEG and JPEG2000.

Manuscript received January 15, 2003; revised August 18, 2003. The work of Z. Wang and E. P. Simoncelli was supported by the Howard Hughes Medical Institute. The work of A. C. Bovik and H. R. Sheikh was supported by the National Science Foundation and the Texas Advanced Research Program. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Reiner Eschbach.

Z. Wang and E. P. Simoncelli are with the Howard Hughes Medical Institute, the Center for Neural Science and the Courant Institute for Mathematical Sciences, New York University, New York, NY 10012 USA (e-mail: zhouwang@ieee.org; eero.simoncelli@nyu.edu).

A. C. Bovik and H. R. Sheikh are with the Laboratory for Image and Video Engineering (LIVE), Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: bovik@ece.utexas.edu; hamid.sheikh@ieee.org).

Digital Object Identifier 10.1109/TIP.2003.819861

[1] A MATLAB implementation of the proposed algorithm is available online at http://www.cns.nyu.edu/~lcv/ssim/.

1057-7149/04$20.00 © 2004 IEEE

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1249
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

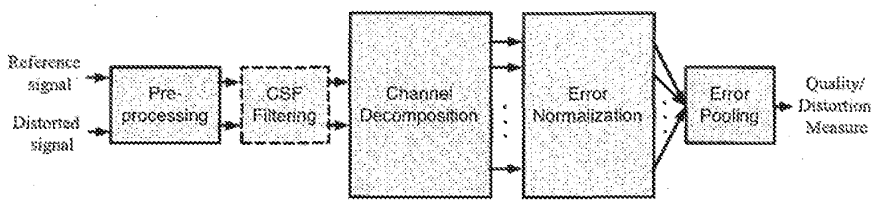


Fig. 1. A prototypical quality assessment system based on error sensitivity. Note that the CSF feature can be implemented either as a separate stage (as shown) or within "Error Normalization."

II. Image Quality Assessment Based on Error Sensitivity

An image signal whose quality is being evaluated can be thought of as a sum of an undistorted reference signal and an error signal. A widely adopted assumption is that the loss of perceptual quality is directly related to the visibility of the error signal. The simplest implementation of this concept is the MSE, which objectively quantifies the strength of the error signal. But two distorted images with the same MSE may have very different types of errors, some of which are much more visible than others. Most perceptual image quality assessment approaches proposed in the literature attempt to weight different aspects of the error signal according to their visibility, as determined by psychophysical measurements in humans or physiological measurements in animals. This approach was pioneered by Mannos and Sakrison [10], and has been extended by many other researchers over the years. Reviews on image and video quality assessment algorithms can be found in [4] and [11]–[13].

A. Framework

Fig. 1 illustrates a generic image quality assessment framework based on error sensitivity. Most perceptual quality assessment models can be described with a similar diagram, although they differ in detail. The stages of the diagram are as follows.

- *Pre-processing*: This stage typically performs a variety of basic operations to eliminate known distortions from the images being compared. First, the distorted and reference signals are properly scaled and aligned. Second, the signal might be transformed into a color space (e.g., [14]) that is more appropriate for the HVS. Third, quality assessment metrics may need to convert the digital pixel values stored in the computer memory into luminance values of pixels on the display device through pointwise nonlinear transformations. Fourth, a low-pass filter simulating the point spread function of the eye optics may be applied. Finally, the reference and the distorted images may be modified using a nonlinear point operation to simulate light adaptation.
- *CSF Filtering*: The contrast sensitivity function (CSF) describes the sensitivity of the HVS to different spatial and temporal frequencies that are present in the visual stimulus. Some image quality metrics include a stage that weights the signal according to this function (typically implemented using a linear filter that approximates the frequency response of the CSF). However, many recent metrics choose to implement CSF as a base-sensitivity normalization factor after channel decomposition.
- *Channel Decomposition*: The images are typically separated into subbands (commonly called "channels" in the psychophysics literature) that are selective for spatial and temporal frequency as well as orientation. While some quality assessment methods implement sophisticated channel decompositions that are believed to be closely related to the neural responses in the primary visual cortex [2], [15]–[19], many metrics use simpler transforms such as the discrete cosine transform (DCT) [20], [21] or separable wavelet transforms [22]–[24]. Channel decompositions tuned to various temporal frequencies have also been reported for video quality assessment [5], [25].
- *Error Normalization*: The error (difference) between the decomposed reference and distorted signals in each channel is calculated and normalized according to a certain masking model, which takes into account the fact that the presence of one image component will decrease the visibility of another image component that is proximate in spatial or temporal location, spatial frequency, or orientation. The normalization mechanism weights the error signal in a channel by a space-varying visibility threshold [26]. The visibility threshold at each point is calculated based on the energy of the reference and/or distorted coefficients in a neighborhood (which may include coefficients from within a spatial neighborhood of the same channel as well as other channels) and the base-sensitivity for that channel. The normalization process is intended to convert the error into units of just noticeable difference (JND). Some methods also consider the effect of contrast response saturation (e.g., [2]).
- *Error Pooling*: The final stage of all quality metrics must combine the normalized error signals over the spatial extent of the image, and across the different channels, into a single value. For most quality assessment methods, pooling takes the form of a Minkowski norm as follows:

$$E(\{e_{l,k}\}) = \left(\sum_{l} \sum_{k} |e_{l,k}|^{\beta} \right)^{\frac{1}{\beta}} \tag{1}$$

where $e_{l,k}$ is the normalized error of the k-th coefficient in the lth channel, and β is a constant exponent typically chosen to lie between 1 and 4. Minkowski pooling may be performed over space (index k) and then over frequency (index l), or vice versa, with some nonlinearity between them, or possibly with different exponents β. A spatial map indicating the relative importance of different regions may also be used to provide spatially variant weighting [25], [27], [28].

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1250
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

B. Limitations

The underlying principle of the error-sensitivity approach is that perceptual quality is best estimated by quantifying the visibility of errors. This is essentially accomplished by simulating the functional properties of early stages of the HVS, as characterized by both psychophysical and physiological experiments. Although this bottom-up approach to the problem has found nearly universal acceptance, it is important to recognize its limitations. In particular, the HVS is a complex and highly nonlinear system, but most models of early vision are based on linear or quasilinear operators that have been characterized using restricted and simplistic stimuli. Thus, error-sensitivity approaches must rely on a number of strong assumptions and generalizations. These have been noted by many previous authors, and we provide only a brief summary here.

- *The Quality Definition Problem*: The most fundamental problem with the traditional approach is the definition of image quality. In particular, it is not clear that error visibility should be equated with loss of quality, as some distortions may be clearly visible but not so objectionable. An obvious example would be multiplication of the image intensities by a global scale factor. The study in [29] also suggested that the correlation between image fidelity and image quality is only moderate.
- *The Suprathreshold Problem*. The psychophysical experiments that underlie many error sensitivity models are specifically designed to estimate the threshold at which a stimulus is just barely visible. These measured threshold values are then used to define visual error sensitivity measures, such as the CSF and various masking effects. However, very few psychophysical studies indicate whether such near-threshold models can be generalized to characterize perceptual distortions significantly larger than threshold levels, as is the case in a majority of image processing situations. In the suprathreshold range, can the relative visual distortions between different channels be normalized using the visibility thresholds? Recent efforts have been made to incorporate suprathreshold psychophysics for analyzing image distortions (e.g., [30]–[34]).
- *The Natural Image Complexity Problem*. Most psychophysical experiments are conducted using relatively simple patterns, such as spots, bars, or sinusoidal gratings. For example, the CSF is typically obtained from threshold experiments using global sinusoidal images. The masking phenomena are usually characterized using a superposition of two (or perhaps a few) different patterns. But all such patterns are much simpler than real world images, which can be thought of as a superposition of a much larger number of simple patterns. Can the models for the interactions between a few simple patterns generalize to evaluate interactions between tens or hundreds of patterns? Is this limited number of simple-stimulus experiments sufficient to build a model that can predict the visual quality of complex-structured natural images? Although the answers to these questions are currently not known, the recently established Modelfest dataset [35] includes both simple and complex patterns, and should facilitate future studies.
- *The Decorrelation Problem*. When one chooses to use a Minkowski metric for spatially pooling errors, one is implicitly assuming that errors at different locations are statistically independent. This would be true if the processing prior to the pooling eliminated dependencies in the input signals. Empirically, however, this is not the case for linear channel decomposition methods such as the wavelet transform. It has been shown that a strong dependency exists between intra- and inter-channel wavelet coefficients of natural images [36], [37]. In fact, state-of-the-art wavelet image compression techniques achieve their success by exploiting this strong dependency [38]–[41]. Psychophysically, various visual masking models have been used to account for the interactions between coefficients [2], [42]. Statistically, it has been shown that a well-designed nonlinear gain control model, in which parameters are optimized to reduce dependencies rather than for fitting data from masking experiments, can greatly reduce the dependencies of the transform coefficients [43], [44]. In [45], [46], it is shown that optimal design of transformation and masking models can reduce both statistical and perceptual dependencies. It remains to be seen how much these models can improve the performance of the current quality assessment algorithms.
- *The Cognitive Interaction Problem*. It is widely known that cognitive understanding and interactive visual processing (e.g., eye movements) influence the perceived quality of images. For example, a human observer will give different quality scores to the same image if s/he is provided with different instructions [4], [30]. Prior information regarding the image content, or attention and fixation, may also affect the evaluation of the image quality [4], [47]. But most image quality metrics do not consider these effects, as they are difficult to quantify and not well understood.

III. Structural-Similarity-Based Image Quality Assessment

Natural image signals are highly structured: their pixels exhibit strong dependencies, especially when they are spatially proximate, and these dependencies carry important information about the structure of the objects in the visual scene. The Minkowski error metric is based on pointwise signal differences, which are independent of the underlying signal structure. Although most quality measures based on error sensitivity decompose image signals using linear transformations, these do not remove the strong dependencies, as discussed in the previous section. The motivation of our new approach is to find a more direct way to compare the structures of the reference and the distorted signals.

A. New Philosophy

In [6] and [9], a new framework for the design of image quality measures was proposed, based on the assumption that the human visual system is highly adapted to extract structural information from the viewing field. It follows that a measure of structural information change can provide a good approximation to perceived image distortion.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1251
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

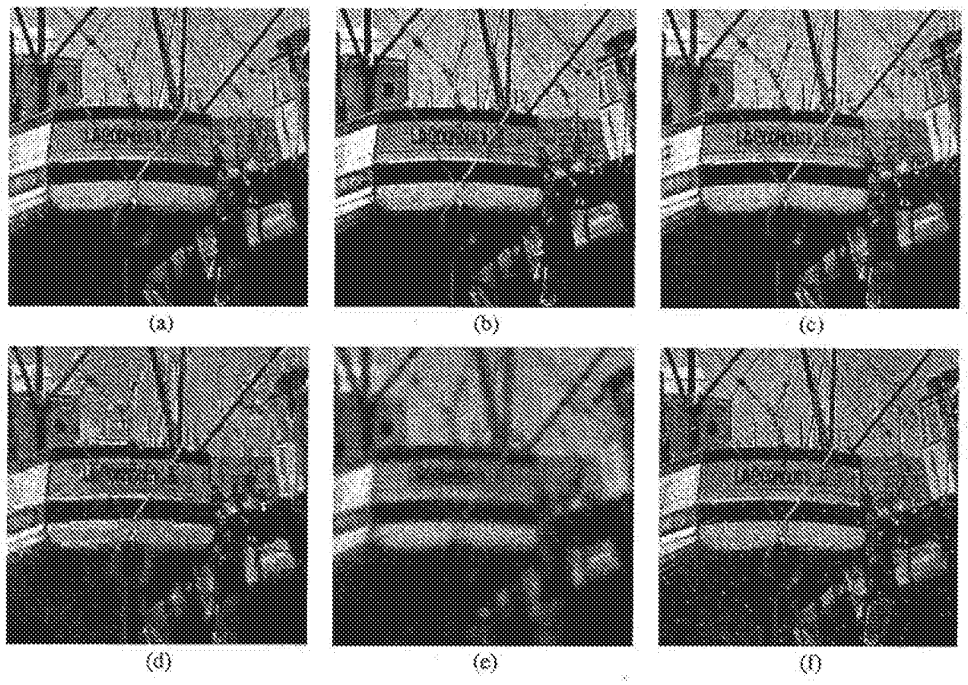


Fig. 2. Comparison of "Boat" images with different types of distortions, all with MSE = 210. (a) Original image (8 bits/pixel; cropped from 512 × 512 to 256 × 256 for visibility). (b) Contrast-stretched image, MSSIM = 0.9168. (c) Mean-shifted image, MSSIM = 0.9900. (d) JPEG compressed image, MSSIM = 0.6949. (e) Blurred image, MSSIM = 0.7052. (f) Salt-pepper impulsive noise contaminated image, MSSIM = 0.7748.

This new philosophy can be best understood through comparison with the error sensitivity philosophy. First, the error sensitivity approach estimates *perceived errors* to quantify image degradations, while the new philosophy considers image degradations as *perceived changes in structural information variation*. A motivating example is shown in Fig. 2, where the original "Boat" image is altered with different distortions, each adjusted to yield nearly identical MSE relative to the original image. Despite this, the images can be seen to have drastically different perceptual quality. With the error sensitivity philosophy, it is difficult to explain why the contrast-stretched image has very high quality in consideration of the fact that its visual difference from the reference image is easily discerned. But it is easily understood with the new philosophy since nearly all the structural information of the reference image is preserved, in the sense that the original information can be nearly fully recovered via a simple pointwise inverse linear luminance transform (except perhaps for the very bright and dark regions where saturation occurs). On the other hand, some structural information from the original image is permanently lost in the JPEG compressed and the blurred images, and therefore they should be given lower quality scores than the contrast-stretched and mean-shifted images.

Second, the error-sensitivity paradigm is a *bottom-up* approach, simulating the function of relevant early-stage components in the HVS. The new paradigm is a *top-down* approach, mimicking the hypothesized functionality of the overall HVS. This, on the one hand, avoids the suprathreshold problem mentioned in the previous section because it does not rely on threshold psychophysics to quantify the perceived distortions. On the other hand, the cognitive interaction problem is also reduced to a certain extent because probing the structures of the objects being observed is thought of as the purpose of the entire process of visual observation, including high level and interactive processes.

Third, the problems of natural image complexity and decorrelation are also avoided to some extent because the new philosophy does not attempt to predict image quality by accumulating the errors associated with psychophysically understood simple patterns. Instead, the new philosophy proposes to evaluate the structural changes between two complex-structured signals directly.

B. The SSIM Index

We construct a specific example of a SSIM quality measure from the perspective of image formation. A previous instantiation of this approach was made in [6]–[8] and promising results on simple tests were achieved. In this paper, we generalize this algorithm, and provide a more extensive set of validation results.

The luminance of the surface of an object being observed is the product of the illumination and the reflectance, but the structures of the objects in the scene are independent of the illumination. Consequently, to explore the structural information in an image, we wish to separate the influence of the illumination. We define the structural information in an image as those attributes that represent the structure of objects in the scene, independent of the average luminance and contrast. Since luminance

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1252
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

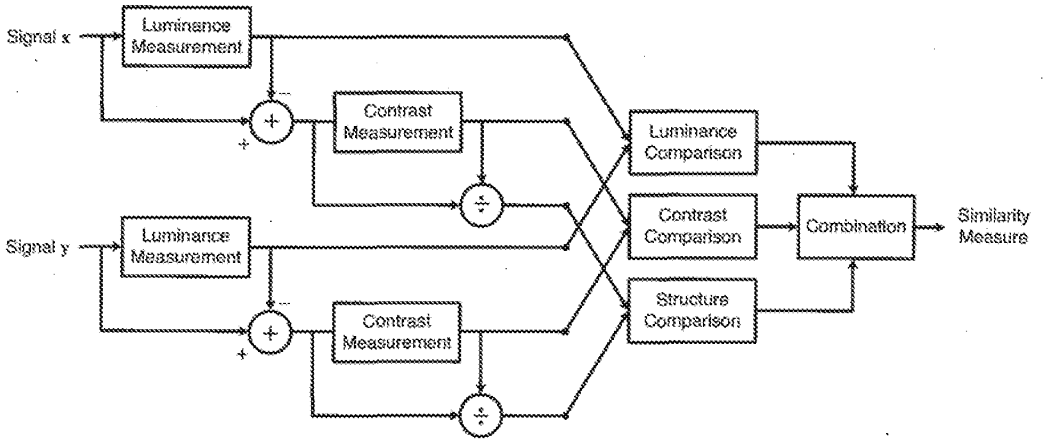


Fig. 3. Diagram of the structural similarity (SSIM) measurement system.

and contrast can vary across a scene, we use the *local* luminance and contrast for our definition.

The system diagram of the proposed quality assessment system is shown in Fig. 3. Suppose $\mathbf{x}$ and $\mathbf{y}$ are two nonnegative image signals, which have been aligned with each other (e.g., spatial patches extracted from each image). If we consider one of the signals to have perfect quality, then the similarity measure can serve as a quantitative measurement of the quality of the second signal. The system separates the task of similarity measurement into three comparisons: luminance, contrast and structure. First, the luminance of each signal is compared. Assuming discrete signals, this is estimated as the mean intensity

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{2}$$

The luminance comparison function $l(\mathbf{x}, \mathbf{y})$ is then a function of μ_x and μ_y.

Second, we remove the mean intensity from the signal. In discrete form, the resulting signal $\mathbf{x} - \mu_x$ corresponds to the projection of vector $\mathbf{x}$ onto the hyperplane defined by

$$\sum_{i=1}^{N} x_i = 0. \tag{3}$$

We use the standard deviation (the square root of variance) as an estimate of the signal contrast. An unbiased estimate in discrete form is given by

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}}. \tag{4}$$

The contrast comparison $c(\mathbf{x}, \mathbf{y})$ is then the comparison of σ_x and σ_y.

Third, the signal is normalized (divided) by its own standard deviation, so that the two signals being compared have unit standard deviation. The structure comparison $s(\mathbf{x}, \mathbf{y})$ is conducted on these normalized signals $(\mathbf{x} - \mu_x)/\sigma_x$ and $(\mathbf{y} - \mu_y)/\sigma_y$.

Finally, the three components are combined to yield an overall similarity measure

$$S(\mathbf{x}, \mathbf{y}) = f\left(l(\mathbf{x}, \mathbf{y}), c(\mathbf{x}, \mathbf{y}), s(\mathbf{x}, \mathbf{y})\right). \tag{5}$$

An important point is that the three components are relatively independent. For example, the change of luminance and/or contrast will not affect the structures of images.

In order to complete the definition of the similarity measure in (5), we need to define the three functions $l(\mathbf{x}, \mathbf{y})$, $c(\mathbf{x}, \mathbf{y})$, and $s(\mathbf{x}, \mathbf{y})$, as well as the combination function $f(\cdot)$. We also would like the similarity measure to satisfy the following conditions.

1) Symmetry: $S(\mathbf{x}, \mathbf{y}) = S(\mathbf{y}, \mathbf{x})$.
2) Boundedness: $S(\mathbf{x}, \mathbf{y}) \leq 1$.
3) Unique maximum: $S(\mathbf{x}, \mathbf{y}) = 1$ if and only if $\mathbf{x} = \mathbf{y}$ (in discrete representations, $x_i = y_i$ for all $i = 1, 2, \cdots, N$).

For luminance comparison, we define

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{6}$$

where the constant C_1 is included to avoid instability when $\mu_x^2 + \mu_y^2$ is very close to zero. Specifically, we choose

$$C_1 = (K_1 L)^2 \tag{7}$$

where L is the dynamic range of the pixel values (255 for 8-bit grayscale images), and $K_1 \ll 1$ is a small constant. Similar considerations also apply to contrast comparison and structure comparison described later. Equation (6) is easily seen to obey the three properties listed above.

Equation (6) is also qualitatively consistent with Weber's law, which has been widely used to model light adaptation (also called luminance masking) in the HVS. According to Weber's law, the magnitude of a just-noticeable luminance change ΔI is approximately proportional to the background luminance I for a wide range of luminance values. In other words, the HVS is sensitive to the *relative* luminance change, and not the absolute luminance change. Letting R represent the size of luminance

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1253
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

change relative to background luminance, we rewrite the luminance of the distorted signal as $\mu_y = (1+R)\mu_x$. Substituting this into (6) gives

$$l(\mathbf{x},\mathbf{y}) = \frac{2(1+R)}{1+(1+R)^2+\frac{C_1}{\mu_x^2}}. \tag{8}$$

If we assume C_1 is small enough (relative to μ_x^2) to be ignored, then $l(\mathbf{x},\mathbf{y})$ is a function only of R, qualitatively consistent with Weber's law.

The contrast comparison function takes a similar form

$$c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2+\sigma_y^2+C_2} \tag{9}$$

where $C_2 = (K_2L)^2$, and $K_2 \ll 1$. This definition again satisfies the three properties listed above. An important feature of this function is that with the same amount of contrast change $\Delta\sigma = \sigma_y - \sigma_x$, this measure is less sensitive to the case of high base contrast σ_x than low base contrast. This is consistent with the contrast-masking feature of the HVS.

Structure comparison is conducted after luminance subtraction and variance normalization. Specifically, we associate the two unit vectors $(\mathbf{x}-\mu_x)/\sigma_x$ and $(\mathbf{y}-\mu_y)/\sigma_y$, each lying in the hyperplane defined by (3), with the structure of the two images. The correlation (inner product) between these is a simple and effective measure to quantify the structural similarity. Notice that the correlation between $(\mathbf{x}-\mu_x)/\sigma_x$ and $(\mathbf{y}-\mu_y)/\sigma_y$ is equivalent to the correlation coefficient between $\mathbf{x}$ and $\mathbf{y}$. Thus, we define the structure comparison function as follows:

$$s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy}+C_3}{\sigma_x\sigma_y+C_3}. \tag{10}$$

As in the luminance and contrast measures, we have introduced a small constant in both denominator and numerator. In discrete form, σ_{xy} can be estimated as

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu_x)(y_i-\mu_y). \tag{11}$$

Geometrically, the correlation coefficient corresponds to the cosine of the angle between the vectors $\mathbf{x}-\mu_x$ and $\mathbf{y}-\mu_y$. Note also that $s(\mathbf{x},\mathbf{y})$ can take on negative values.

Finally, we combine the three comparisons of (6), (9) and (10) and name the resulting similarity measure the SSIM index between signals $\mathbf{x}$ and $\mathbf{y}$

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^\alpha \cdot [c(\mathbf{x},\mathbf{y})]^\beta \cdot [s(\mathbf{x},\mathbf{y})]^\gamma \tag{12}$$

where $\alpha > 0$, $\beta > 0$ and $\gamma > 0$ are parameters used to adjust the relative importance of the three components. It is easy to verify that this definition satisfies the three conditions given above. In order to simplify the expression, we set $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$ in this paper. This results in a specific form of the SSIM index

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y+C_1)(2\sigma_{xy}+C_2)}{(\mu_x^2+\mu_y^2+C_1)(\sigma_x^2+\sigma_y^2+C_2)}. \tag{13}$$

The "universal quality index" (UQI) defined in [6] and [7] corresponds to the special case that $C_1 = C_2 = 0$, which produces unstable results when either $(\mu_x^2+\mu_y^2)$ or $(\sigma_x^2+\sigma_y^2)$ is very close to zero.

The relationship between the SSIM index and more traditional quality metrics may be illustrated geometrically in a vector space of image components. These image components can be either pixel intensities or other extracted features such as transformed linear coefficients. Fig. 4 shows equal-distortion contours drawn around three different example reference vectors, each of which represents the local content of one reference image. For the purpose of illustration, we show only a two-dimensional space, but in general the dimensionality should match the number of image components being compared. Each contour represents a set of images with equal distortions relative to the enclosed reference image. Fig. 4(a) shows the result for a simple Minkowski metric. Each contour has the same size and shape (a circle here, as we are assuming an exponent of 2). That is, perceptual distance corresponds to Euclidean distance. Fig. 4(b) shows a Minkowski metric in which different image components are weighted differently. This could be, for example, weighting according to the CSF, as is common in many models. Here the contours are ellipses, but still are all the same size. These are shown aligned with the axes, but in general could be tilted to any fixed orientation.

Many recent models incorporate contrast masking behaviors, which has the effect of rescaling the equal-distortion contours according to the signal magnitude, as shown in Fig. 4(c). This may be viewed as a type of *adaptive* distortion metric: it depends not just on the difference between the signals, but also on the signals themselves. Fig. 4(d) shows a combination of contrast masking (magnitude weighting) followed by component weighting. Our proposed method, on the other hand, separately computes a comparison of two independent quantities: the vector lengths, and their angles. Thus, the contours will be aligned with the axes of a polar coordinate system. Figs. 4(e) and 4(f) show two examples of this, computed with different exponents. Again, this may be viewed as an *adaptive* distortion metric, but unlike previous models, both the size and the shape of the contours are adapted to the underlying signal. Some recent models that use divisive normalization to describe masking effects also exhibit signal-dependent contour orientations (e.g., [45], [46], [48]), although precise alignment with the axes of a polar coordinate system as in Fig. 4(e) and (f) is not observed in these methods.

C. Image Quality Assessment Using SSIM Index

For image quality assessment, it is useful to apply the SSIM index locally rather than globally. First, image statistical features are usually highly spatially nonstationary. Second, image distortions, which may or may not depend on the local image statistics, may also be space-variant. Third, at typical viewing distances, only a local area in the image can be perceived with high resolution by the human observer at one time instance (because of the foveation feature of the HVS [49], [50]). And finally, localized quality measurement can provide a spatially varying quality map of the image, which delivers more information about the quality degradation of the image and may be useful in some applications.

In [6] and [7], the local statistics μ_x, σ_x and σ_{xy} are computed within a local 8×8 square window, which moves pixel-by-pixel over the entire image. At each step, the local statistics and SSIM index are calculated within the local window. One problem with

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1254
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

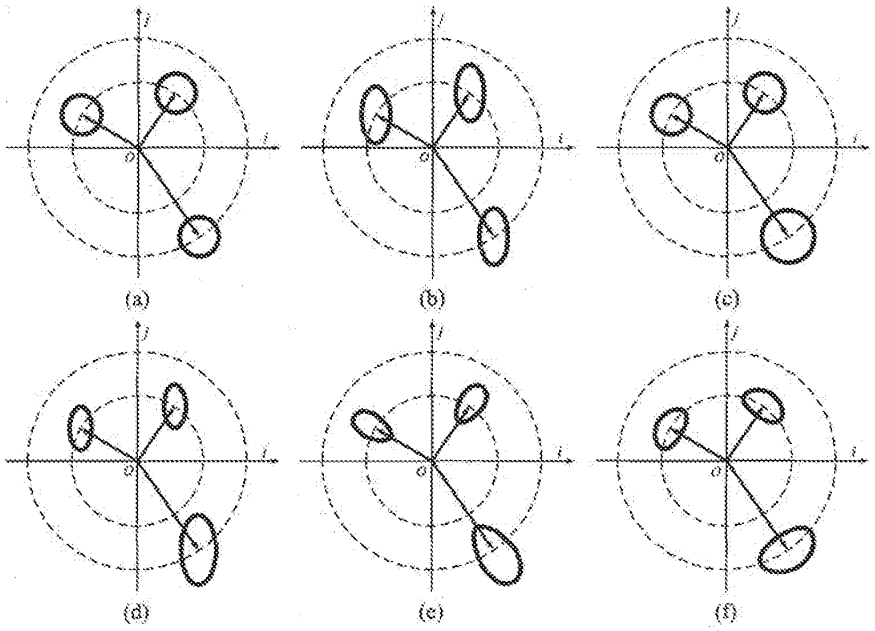


Fig. 4. Three example equal-distance contours for different quality metrics. (a) Minkowski error measurement systems. (b) Component weighted Minkowski error measurement systems. (c) Magnitude-weighted Minkowski error measurement systems. (d) Magnitude and component-weighted Minkowski error measurement systems. (e) The proposed system (a combination of (9) and (10)) with more emphasis on $s(\mathbf{x}, \mathbf{y})$. (f) The proposed system [a combination of (9) and (10)] with more emphasis on $c(\mathbf{x}, \mathbf{y})$. Each image is represented as a vector, whose entries are image components. Note: this is an illustration in 2-D space. In practice, the number of dimensions should be equal to the number of image components used for comparison (e.g., the number of pixels or transform coefficients).

this method is that the resulting SSIM index map often exhibits undesirable "blocking" artifacts. In this paper, we use an 11×11 circular-symmetric Gaussian weighting function $\mathbf{w} = \{w_i | i = 1, 2, \cdots, N\}$, with standard deviation of 1.5 samples, normalized to unit sum ($\sum_{i=1}^{N} w_i = 1$). The estimates of local statistics μ_x, σ_x and σ_{xy} are then modified accordingly as

$$\mu_x = \sum_{i=1}^{N} w_i x_i \tag{14}$$

$$\sigma_x = \left(\sum_{i=1}^{N} w_i (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \tag{15}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y). \tag{16}$$

With such a windowing approach, the quality maps exhibit a locally isotropic property. Throughout this paper, the SSIM measure uses the following parameter settings: $K_1 = 0.01$; $K_2 = 0.03$. These values are somewhat arbitrary, but we find that in our current experiments, the performance of the SSIM index algorithm is fairly insensitive to variations of these values.

In practice, one usually requires a single overall quality measure of the entire image. We use a mean SSIM (MSSIM) index to evaluate the overall image quality

$$\mathrm{MSSIM}(\mathbf{X}, \mathbf{Y}) = \frac{1}{M} \sum_{j=1}^{M} \mathrm{SSIM}(\mathbf{x}_j, \mathbf{y}_j) \tag{17}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are the reference and the distorted images, respectively; $\mathbf{x}_j$ and $\mathbf{y}_j$ are the image contents at the jth local window; and M is the number of local windows of the image. Depending on the application, it is also possible to compute a weighted average of the different samples in the SSIM index map. For example, region-of-interest image processing systems may give different weights to different segmented regions in the image. As another example, it has been observed that different image textures attract human fixations with varying degrees (e.g., [51], [52]). A smoothly varying foveated weighting model (e.g., [50]) can be employed to define the weights. In this paper, however, we use uniform weighting. A MATLAB implementation of the SSIM index algorithm is available online at [53].

IV. Experimental Results

Many image quality assessment algorithms have been shown to behave consistently when applied to distorted images created from the same original image, using the same type of distortions (e.g., JPEG compression). However, the effectiveness of these models degrades significantly when applied to a set of images originating from different reference images, and/or including a variety of different types of distortions. Thus, cross-image and cross-distortion tests are critical in evaluating the effectiveness of an image quality metric. It is impossible to show a thorough set of such examples, but the images in Fig. 2 provide an encouraging starting point for testing the cross-distortion capability of the quality assessment algorithms. The MSE and MSSIM measurement results are given in the figure caption. Obviously, MSE

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1255
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

performs very poorly in this case. The MSSIM values exhibit much better consistency with the qualitative visual appearance.

A. Best-Case/Worst-Case Validation

We also have developed a more efficient methodology for examining the relationship between our objective measure and perceived quality. Starting from a distorted image, we ascend/descend the gradient of MSSIM while constraining the MSE to remain equal to that of the initial distorted image. Specifically, we iterate the following two linear-algebraic steps:

$$(1)\ \mathbf{Y} \leftarrow \mathbf{Y} \pm \lambda P(\mathbf{X}, \mathbf{Y}) \vec{\nabla}_{\mathbf{Y}} \mathrm{MSSIM}(\mathbf{X}, \mathbf{Y})$$
$$(2)\ \mathbf{Y} \leftarrow \mathbf{X} + \sigma \hat{E}(\mathbf{X}, \mathbf{Y})$$

where σ is the square root of the constrained MSE, λ controls the step size, and $\hat{E}(\mathbf{X}, \mathbf{Y})$ is a unit vector defined by

$$\hat{E}(\mathbf{X}, \mathbf{Y}) = \frac{\mathbf{Y} - \mathbf{X}}{\|\mathbf{Y} - \mathbf{X}\|}$$

and $P(\mathbf{X}, \mathbf{Y})$ is a projection operator

$$P(\mathbf{X}, \mathbf{Y}) = \mathbf{I} - \hat{E}(\mathbf{X}, \mathbf{Y}) \hat{E}^T(\mathbf{X}, \mathbf{Y})$$

with $\mathbf{I}$ the identity operator. MSSIM is differentiable and this procedure converges to a local maximum/minimum of the objective measure. Visual inspection of these best- and worst-case images, along with the initial distorted image, provides a visual indication of the types of distortion deemed least/most important by the objective measure. Therefore, it is an expedient and direct method for revealing perceptual implications of the quality measure. An example is shown in Fig. 5, where the initial image is contaminated with Gaussian white noise. It can be seen that the local structures of the original image are very well preserved in the maximal MSSIM image. On the other hand, the image structures are changed dramatically in the worst-case MSSIM image, in some cases reversing contrast.

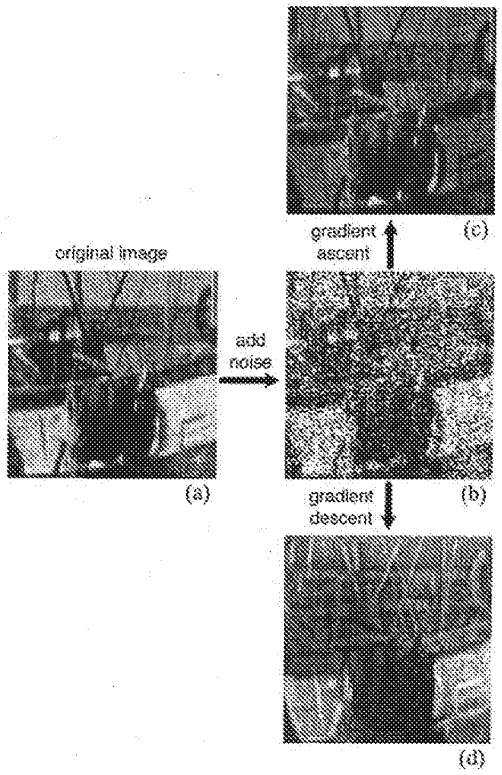


Fig. 5. Best- and worst-case SSIM images, with identical MSE. These are computed by gradient ascent/descent iterative search on MSSIM measure, under the constraint of fixed MSE = 2500. (a) Original image (100 × 100, 8 bits/pixel, cropped from the "Boat" image). (b) Initial image, contaminated with Gaussian white noise (MSSIM = 0.3021). (c) Maximum MSSIM image (MSSIM = 0.9337). (d) Minimum MSSIM image (MSSIM = −0.5411).

B. Test on JPEG and JPEG2000 Image Database

We compare the cross-distortion and cross-image performances of different quality assessment models on an image database composed of JPEG and JPEG2000 compressed images. Twenty-nine high-resolution 24 bits/pixel RGB color images (typically 768 × 512 or similar size) were compressed at a range of quality levels using either JPEG or JPEG2000, producing a total of 175 JPEG images and 169 JPEG2000 images. The bit rates were in the range of 0.150 to 3.336 and 0.028 to 3.150 bits/pixel, respectively, and were chosen nonuniformly such that the resulting distribution of subjective quality scores was approximately uniform over the entire range. Subjects viewed the images from comfortable seating distances (this distance was only moderately controlled, to allow the data to reflect natural viewing conditions), and were asked to provide their perception of quality on a continuous linear scale that was divided into five equal regions marked with adjectives "Bad," "Poor," "Fair," "Good," and "Excellent." Each JPEG and JPEG2000 compressed image was viewed by 13 ∼ 20 subjects and 25 subjects, respectively. The subjects were mostly male college students.

Raw scores for each subject were normalized by the mean and variance of scores for that subject (i.e., raw values were converted to Z-scores [54]) and then the entire data set was rescaled to fill the range from 1 to 100. Mean opinion scores (MOSs) were then computed for each image, after removing outliers (most subjects had no outliers). The average standard deviations (for each image) of the subjective scores for JPEG, JPEG2000, and all images were 6.00, 7.33, and 6.65, respectively. The image database, together with the subjective score and standard deviation for each image, has been made available on the Internet at [55].

The luminance component of each JPEG and JPEG2000 compressed image is averaged over local 2 × 2 window and downsampled by a factor of 2 before the MSSIM value is calculated. Our experiments with the current dataset show that the use of the other color components does not significantly change the performance of the model, though this should not be considered generally true for color image quality assessment. Unlike many other perceptual image quality assessment approaches, no specific training procedure is employed before applying the proposed algorithm to the database, because the proposed method is intended for general-purpose image quality assessment (as opposed to image compression alone).

Figs. 6 and 7 show some example images from the database at different quality levels, together with their SSIM index maps

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1256
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

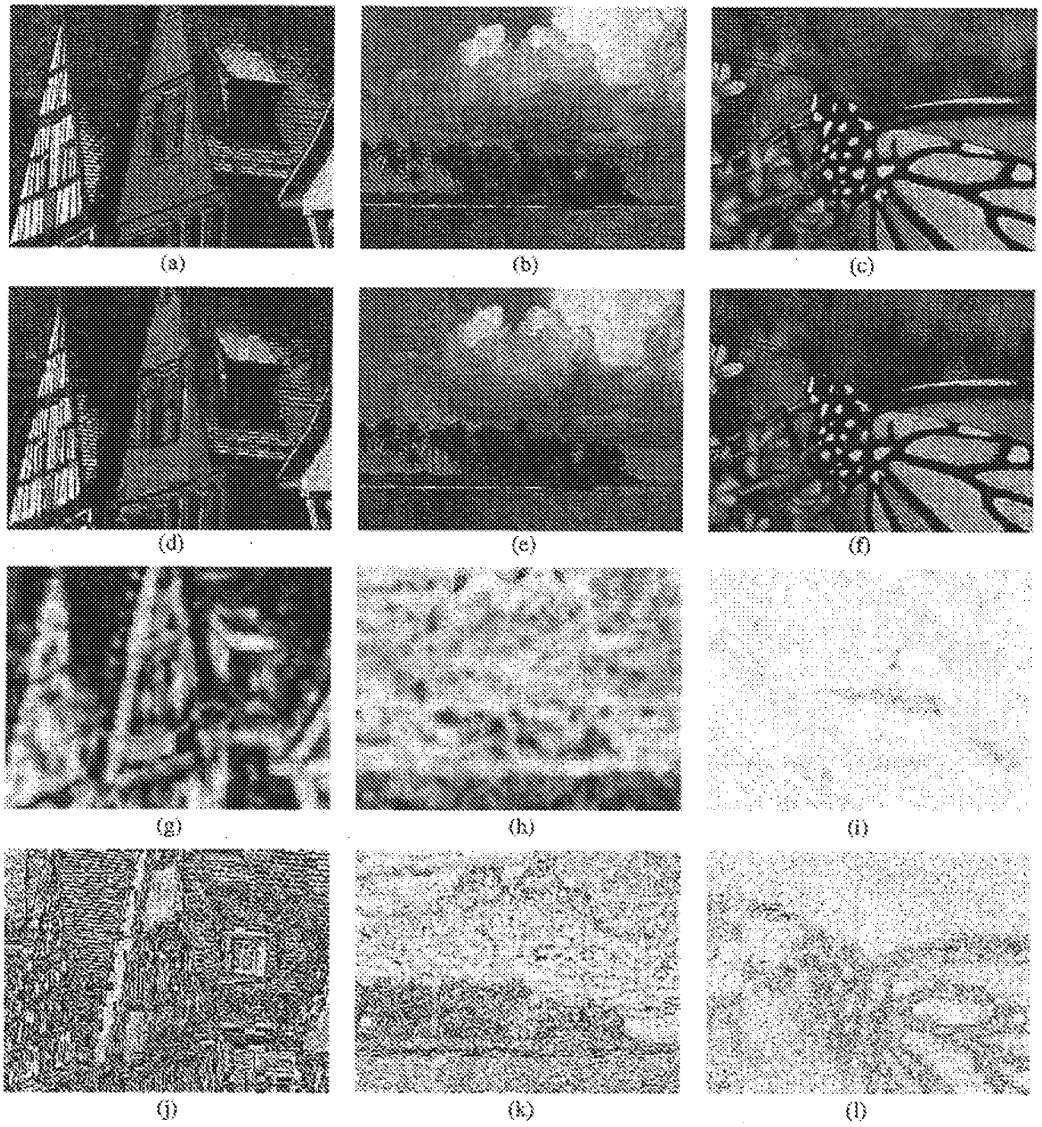


Fig. 6. Sample JPEG images compressed to different quality levels (original size: 768 × 512; cropped to 256 × 192 for visibility). The original (a) "Buildings," (b) "Ocean," and (c) "Monarch" images. (d) Compressed to 0.2673 bits/pixel, PSNR = 21.98 dB, MSSIM = 0.7118. (e) Compressed to 0.2980 bits/pixel, PSNR = 30.87 dB, MSSIM = 0.8886. (f) Compressed to 0.7755 bits/pixel, PSNR = 36.78 dB, MSSIM = 0.9898. (g), (h) and (i) show SSIM maps of the compressed images, where brightness indicates the magnitude of the local SSIM index (squared for visibility). (j), (k) and (l) show absolute error maps of the compressed images (contrast-inverted for easier comparison to the SSIM maps).

and absolute error maps. Note that at low bit rate, the coarse quantization in JPEG and JPEG2000 algorithms often results in smooth representations of fine-detail regions in the image [e.g., the tiles in Fig. 6(d) and the trees in Fig. 7(d)]. Compared with other types of regions, these regions may not be worse in terms of pointwise difference measures such as the absolute error. However, since the structural information of the image details are nearly completely lost, they exhibit poorer visual quality. Comparing Fig. 6(g) with Fig. 6(j), and Fig. 7(g) with 6(j), we observe that the SSIM index is better in capturing such poor quality regions. Also notice that for images with intensive strong edge structures such as Fig. 7(c), it is difficult to reduce the pointwise errors in the compressed image, even at relatively high bit rate, as exemplified by Fig. 7(i). However, the compressed image supplies acceptable perceived quality as shown in Fig. 7(f). In fact, although the visual quality of Fig. 7(f) is better than Fig. 7(e), its absolute error map Fig. 7(l) appears to be worse than Fig. 7(k), as is confirmed by their PSNR values. The SSIM index maps Figs. 7(h) and 7(i) deliver better consistency with perceived quality measurement.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1257
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

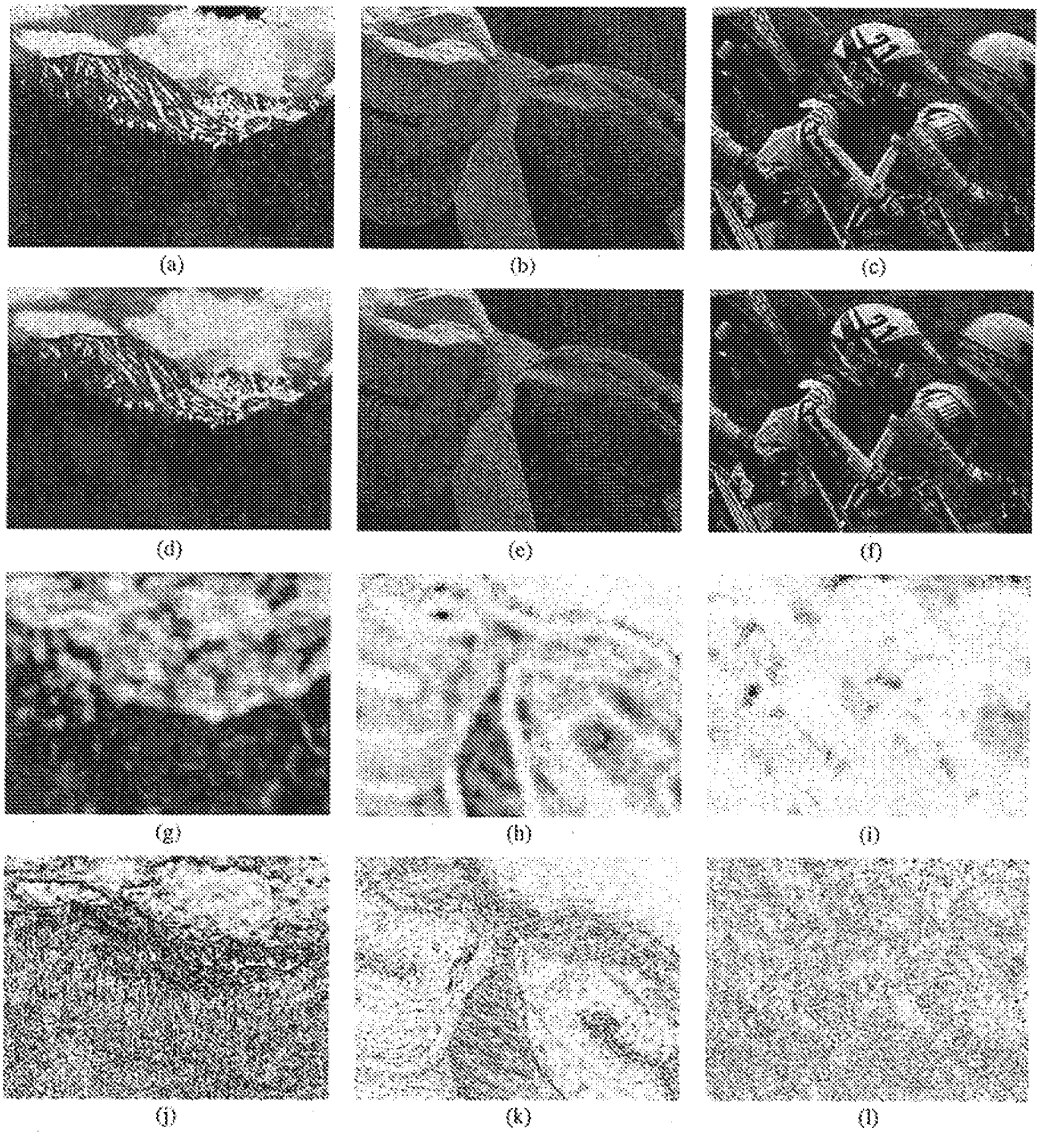


Fig. 7. Sample JPEG2000 images compressed to different quality levels (original size: 768 × 512, cropped to 256 × 192 for visibility). The original (a) "Stream," (b) "Caps," and (c) "Bikes" images, respectively. (d) Compressed to 0.1896 bits/pixel, PSNR = 23.46 dB, MSSIM = 0.7339. (e) Compressed to 0.1982 bits/pixel, PSNR = 34.56 dB, MSSIM = 0.9409. (f) Compressed to 1.1454 bits/pixel, PSNR = 33.47 dB, MSSIM = 0.9747. (g), (h) and (i) show SSIM maps of the compressed images, where brightness indicates the magnitude of the local SSIM index (squared for visibility). (j), (k) and (l) show absolute error maps of the compressed images (contrast-inverted for easier comparison to the SSIM maps).

The quality assessment models used for comparison include PSNR, the well-known Sarnoff model,[2] UQI [7] and MSSIM. The scatter plot of MOS versus model prediction for each model is shown in Fig. 8. If PSNR is considered as a benchmark method to evaluate the effectiveness of the other image quality metrics, the Sarnoff model performs quite well in this test. This is in contrast with previous published test results (e.g., [57], [58]), where the performance of most models (including the Sarnoff model) were reported to be statistically equivalent to root mean squared error [57] and PSNR [58]. The UQI method performs much better than MSE for the simple cross-distortion test in [7], [8], but does not deliver satisfactory results in Fig. 8. We think the major reason is that at nearly flat regions, the denominator of the contrast comparison formula is close to zero, which makes the algorithm unstable. By inserting the small constants C_1 and C_2, MSSIM completely avoids this problem and the scatter slot demonstrates that it supplies remarkably good prediction of the subjective scores.

[2]Available at http://www.sarnoff.com/products_services/video_vision/jndmetrix/.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1258
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

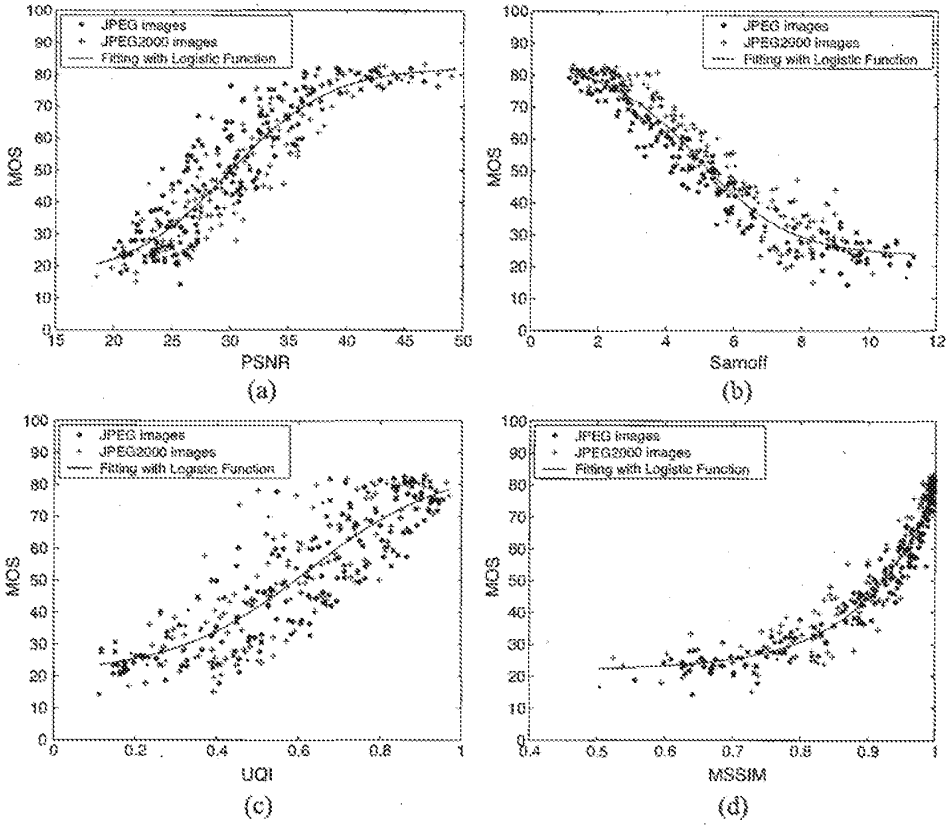


Fig. 8. Scatter plots of subjective mean opinion score (MOS) versus model prediction. Each sample point represents one test image. (a) PSNR. (b) Sarnoff model (using Sarnoff JNDmetrix 8.0 [55]). (c) UQI [7] (equivalent to MSSIM with square window and $K_1 = K_2 = 0$). d) MSSIM (Gaussian window, $K_1 = 0.01$; $K_2 = 0.03$).

In order to provide quantitative measures on the performance of the objective quality assessment models, we follow the performance evaluation procedures employed in the video quality experts group (VQEG) Phase I FR-TV test [58], where four evaluation metrics were used. First, logistic functions are used in a fitting procedure to provide a nonlinear mapping between the objective/subjective scores. The fitted curves are shown in Fig. 8. In [58], Metric 1 is the correlation coefficient between objective/subjective scores after variance-weighted regression analysis. Metric 2 is the correlation coefficient between objective/subjective scores after nonlinear regression analysis. These two metrics combined, provide an evaluation of *prediction accuracy*. The third metric is the Spearman rank-order correlation coefficient between the objective/subjective scores. It is considered as a measure of *prediction monotonicity*. Finally, metric 4 is the outlier ratio (percentage of the number of predictions outside the range of ± 2 times of the standard deviations) of the predictions after the nonlinear mapping, which is a measure of *prediction consistency*. More details on these metrics can be found in [58]. In addition to these, we also calculated the mean absolute prediction error (MAE), and root mean square prediction error (rms) after nonlinear regression, and weighted mean absolute prediction error (WMAE) and weighted root mean square prediction error (WRMS) after variance-weighted regression. The evaluation results for all the models being compared are given in Table I. For every one of these criteria, MSSIM performs better than all of the other models being compared.

V. Discussion

In this paper, we have summarized the traditional approach to image quality assessment based on error-sensitivity, and have enumerated its limitations. We have proposed the use of structural similarity as an alternative motivating principle for the design of image quality measures. To demonstrate our structural similarity concept, we developed an SSIM index and showed that it compares favorably with other methods in accounting for our experimental measurements of subjective quality of 344 JPEG and JPEG2000 compressed images.

Although the proposed SSIM index method is motivated from substantially different design principles, we see it as complementary to the traditional approach. Careful analysis shows that both the SSIM index and several recently developed divisive-normalization based masking models exhibit input-dependent behavior in measuring signal distortions [45], [46], [48]. It seems possible that the two approaches may eventually converge to similar solutions.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1259
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

TABLE I
PERFORMANCE COMPARISON OF IMAGE QUALITY ASSESSMENT MODELS. CC: CORRELATION COEFFICIENT; MAE: MEAN ABSOLUTE ERROR; RMS: ROOT MEAN SQUARED ERROR; OR: OUTLIER RATIO; WMAE: WEIGHTED MEAN ABSOLUTE ERROR; WRMS: WEIGHTED ROOT MEAN SQUARED ERROR; SROCC: SPEARMAN RANK-ORDER CORRELATION COEFFICIENT

	Non-linear Regression				Variance-weighted Regression				Rank-order
Model	CC	MAE	RMS	OR	CC	WMAE	WRMS	OR	SROCC
PSNR	0.905	6.53	8.46	0.157	0.903	6.18	8.26	0.140	0.901
Sarnoff	0.956	4.66	5.81	0.064	0.956	4.42	5.62	0.061	0.947
UQI	0.866	7.76	9.90	0.169	0.861	7.64	9.79	0.196	0.865
MSSIM	0.967	3.95	5.06	0.041	0.967	3.79	4.87	0.041	0.963

There are a number of issues that are worth investigation with regard to the specific SSIM index of (12). First, the optimization of the SSIM index for various image processing algorithms needs to be studied. For example, it may be employed for rate-distortion optimizations in the design of image compression algorithms. This is not an easy task since (12) is mathematically more cumbersome than MSE. Second, the application scope of the SSIM index may not be restricted to image processing. In fact, because it is a symmetric measure, it can be thought of as a similarity measure for comparing any two signals. The signals can be either discrete or continuous, and can live in a space of arbitrary dimensionality.

We consider the proposed SSIM indexing approach as a particular implementation of the philosophy of structural similarity, from an image formation point of view. Under the same philosophy, other approaches may emerge that could be significantly different from the proposed SSIM indexing algorithm. Creative investigation of the concepts of structural information and structural distortion are likely to drive the success of these innovations.

ACKNOWLEDGMENT

The authors would like to thank Dr. J. Malo and Dr. Lu for insightful comments, Dr. J. Lubin and Dr. D. Dixon for providing the Sarnoff JNDmetrix software, Dr. P. Corriveau and Dr. J. Libert for supplying the routines used in VQEG Phase I FR-TV test for the regression analysis of subjective/objective data comparison, and Visual Delights, Inc. for allowing the authors to use their images for subjective experiments.

REFERENCES

[1] B. Girod, "What's wrong with mean-squared error," in *Digital Images and Human Vision*, A. B. Watson, Ed. Cambridge, MA: MIT Press, 1993, pp. 207–220.
[2] P. C. Teo and D. J. Heeger, "Perceptual image distortion," in *Proc. SPIE*, vol. 2179, 1994, pp. 127–141.
[3] A. M. Eskicioglu and P. S. Fisher, "Image quality measures and their performance," *IEEE Trans. Commun.*, vol. 43, pp. 2959–2965, Dec. 1995.
[4] M. P. Eckert and A. P. Bradley, "Perceptual quality metrics applied to still image compression," *Signal Processing*, vol. 70, pp. 177–200, Nov. 1998.
[5] S. Winkler, "A perceptual distortion metric for digital color video," in *Proc. SPIE*, vol. 3644, 1999, pp. 175–184.
[6] Z. Wang, "Rate scalable Foveated image and video communications," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Texas at Austin, Austin, TX, Dec. 2001.
[7] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, pp. 81–84, Mar. 2002.
[8] Z. Wang, Demo Images and Free Software for 'a Universal Image Quality Index'. [Online] Available: http://anchovy.ece.utexas.edu/~zwang/research/quality_index/demo.html
[9] Z. Wang, A. C. Bovik, and L. Lu, "Why is image quality assessment so difficult," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 4, Orlando, FL, May 2002, pp. 3313–3316.
[10] J. L. Mannos and D. J. Sakrison, "The effects of a visual fidelity criterion on the encoding of images," *IEEE Trans. Inform. Theory*, vol. IT-4, pp. 525–536, 1974.
[11] T. N. Pappas and R. J. Safranek, "Perceptual criteria for image quality evaluation," in *Handbook of Image and Video Proc.*, A. Bovik, Ed. New York: Academic, 2000.
[12] Z. Wang, H. R. Sheikh, and A. C. Bovik, "Objective video quality assessment," in *The Handbook of Video Databases: Design and Applications*, B. Furht and O. Marques, Eds. Boca Raton, FL: CRC Press, 2003.
[13] S. Winkler, "Issues in vision modeling for perceptual video quality assessment," *Signal Processing*, vol. 78, pp. 231–252, 1999.
[14] A. B. Poirson and B. A. Wandell, "Appearance of colored patterns: pattern-color separability," *J. Opt. Soc. Amer. A: Opt. Image Sci.*, vol. 10, no. 12, pp. 2458–2470, 1993.
[15] A. B. Watson, "The cortex transform: rapid computation of simulated neural images," *Comput. Vis., Graph., Image Process.*, vol. 39, pp. 311–327, 1987.
[16] S. Daly, "The visible difference predictor: an algorithm for the assessment of image fidelity," in *Digital Images and Human Vision*, A. B. Watson, Ed. Cambridge, MA: MIT Press, 1993, pp. 179–206.
[17] J. Lubin, "The use of psychophysical data and models in the analysis of display system performance," in *Digital Images and Human Vision*, A. B. Watson, Ed. Cambridge, MA: MIT Press, 1993, pp. 163–178.
[18] D. J. Heeger and P. C. Teo, "A model of perceptual image fidelity," in *Proc. IEEE Int. Conf. Image Processing*, 1995, pp. 343–345.
[19] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, "Shiftable multi-scale transforms," *IEEE Trans. Inform. Theory*, vol. 38, pp. 587–607, 1992.
[20] A. B. Watson, "DCT quantization matrices visually optimized for individual images," in *Proc. SPIE*, vol. 1913, 1993.
[21] A. B. Watson, J. Hu, and J. F. McGowan III, "DVQ: a digital video quality metric based on human vision," *J. Electron. Imaging*, vol. 10, no. 1, pp. 20–29, 2001.
[22] A. B. Watson, G. Y. Yang, J. A. Solomon, and J. Villasenor, "Visibility of wavelet quantization noise," *IEEE Trans. Image Processing*, vol. 6, pp. 1164–1175, Aug. 1997.
[23] A. P. Bradley, "A wavelet visible difference predictor," *IEEE Trans. Image Processing*, vol. 5, pp. 717–730, May 1999.
[24] Y. K. Lai and C.-C. J. Kuo, "A Haar wavelet approach to compressed image quality measurement," *J. Vis. Commun. Image Repres.*, vol. 11, pp. 17–40, Mar. 2000.
[25] C. J. van den Branden Lambrecht and O. Verscheure, "Perceptual quality measure using a spatio-temporal model of the human visual system," in *Proc. SPIE*, vol. 2668, 1996, pp. 450–461.
[26] A. B. Watson and J. A. Solomon, "Model of visual contrast gain control and pattern masking," *J. Opt. Soc. Amer.*, vol. 14, no. 9, pp. 2379–2391, 1997.
[27] W. Xu and G. Hauske, "Picture quality evaluation based on error segmentation," in *Proc. SPIE*, vol. 2308, 1994, pp. 1454–1465.
[28] W. Osberger, N. Bergmann, and A. Maeder, "An automatic image quality assessment technique incorporating high level perceptual factors," in *Proc. IEEE Int. Conf. Image Processing*, 1998, pp. 414–418.
[29] D. A. Silverstein and J. E. Farrell, "The relationship between image fidelity and image quality," in *Proc. IEEE Int. Conf. Image Processing*, 1996, pp. 881–884.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1260
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[30] D. R. Fuhrmann, J. A. Baro, and J. R. Cox Jr., "Experimental evaluation of psychophysical distortion metrics for JPEG-encoded images," *J. Electron. Imaging*, vol. 4, pp. 397–406, Oct. 1995.
[31] A. B. Watson and L. Kreslake, "Measurement of visual impairment scales for digital video," in *Proc. SPIE—Human Vision, Visual Processing, and Digital Display.*, vol. 4299, 2001.
[32] J. G. Ramos and S. S. Hemami, "Suprathreshold wavelet coefficient quantization in complex stimuli: psychophysical evaluation and analysis," *J. Opt. Soc. Amer. A*, vol. 18, pp. 2385–2397, 2001.
[33] D. M. Chandler and S. S. Hemami, "Additivity models for suprathreshold distortion in quantized wavelet-coded images," in *Proc. SPIE—Human Vision and Electronic Imaging VII*, vol. 4662, Jan. 2002.
[34] J. Xing, "An image processing model of contrast perception and discrimination of the human visual system," in *SID Conf.*, Boston, MA, May 2002.
[35] A. B. Watson, "Visual detection of spatial contrast patterns: evaluation of five simple models," *Opt. Exp.*, vol. 6, pp. 12–33, Jan. 2000.
[36] E. P. Simoncelli, "Statistical models for images: compression, restoration and synthesis," in *Proc 31st Asilomar Conf. Signals, Systems and Computers*, Pacific Grove, CA, Nov. 1997, pp. 673–678.
[37] J. Liu and P. Moulin, "Information-theoretic analysis of interscale and intrascale dependencies between image wavelet coefficients," *IEEE Trans. Image Processing*, vol. 10, pp. 1647–1658, Nov. 2001.
[38] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, pp. 3445–3462, Dec. 1993.
[39] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, June 1996.
[40] R. W. Buccigrossi and E. P. Simoncelli, "Image compression via joint statistical characterization in the wavelet domain," *IEEE Trans. Image Processing*, vol. 8, pp. 1688–1701, Dec. 1999.
[41] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards, and Practice*. Norwell, MA: Kluwer, 2001.
[42] J. M. Foley and G. M. Boynton, "A new model of human luminance pattern vision mechanisms: analysis of the effects of pattern orientation, spatial phase, and temporal frequency," in *Proc. SPIE—Computational Vision Based on Neurobiology*, vol. 2054, T. A. Lawton, Ed., 1994.
[43] O. Schwartz and E. P. Simoncelli, "Natural signal statistics and sensory gain control," *Nature: Neuroscience*, vol. 4, pp. 819–825, Aug. 2001.
[44] M. J. Wainwright, O. Schwartz, and E. P. Simoncelli, "Natural image statistics and divisive normalization: modeling nonlinearity and adaptation in cortical neurons," in *Probabilistic Models of the Brain: Perception and Neural Function*, R. Rao, B. Olshausen, and M. Lewicki, Eds. Cambridge, MA: MIT Press, 2002.
[45] J. Malo, R. Navarro, I. Epifanio, F. Ferri, and J. M. Artigas, "Non-linear invertible representation for joint statistical and perceptual feature decorrelation," *Lecture Notes on Computer Science*, vol. 1876, pp. 658–667, 2000.
[46] I. Epifanio, J. Gutiérrez, and J. Malo, "Linear transform for simultaneous diagonalization of covariance and perceptual metric matrix in image coding," *Pattern Recognit.*, vol. 36, pp. 1799–1811, Aug. 2003.
[47] W. F. Good, G. S. Maitz, and D. Gur, "Joint photographic experts group (JPEG) compatible data compression of mammograms," *J. Dig. Imaging*, vol. 17, no. 3, pp. 123–132, 1994.
[48] A. Puns, J. Malo, J. M. Artigas, and P. Capilla, "Image quality metric based on multidimensional contrast perception models," *Displays*, vol. 20, pp. 93–110, 1999.
[49] W. S. Geisler and M. S. Banks, "Visual performance," in *Handbook of Optics*, M. Bass, Ed. New York: McGraw-Hill, 1995.
[50] Z. Wang and A. C. Bovik, "Embedded foveation image coding," *IEEE Trans. Image Processing*, vol. 10, pp. 1397–1410, Oct. 2001.
[51] C. M. Privitera and L. W. Stark, "Algorithms for defining visual regions-of-interest: comparison with eye fixations," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 970–982, Sept. 2000.
[52] U. Rajashekar, L. K. Cormack, and A. C. Bovik, "Image features that draw fixations," in *Proc. IEEE Int. Conf. Image Processing*, Barcelona, Spain, Sept. 2003.
[53] Z. Wang, The SSIM Index for Image Quality Assessment. [Online] Available: http://www.cns.nyu.edu/~lcv/ssim/
[54] A. M. van Dijk, J. B. Martens, and A. B. Watson, "Quality assessment of coded images using numerical category scaling," in *Proc. SPIE*, vol. 2451, 1995.
[55] H. R. Sheikh, Z. Wang, A. C. Bovik, and L. K. Cormack, Image and Video Quality Assessment Research at LIVE. [Online] Available: http://live.ece.utexas.edu/research/quality/
[56] J. Lubin, "A visual discrimination model for imaging system design and evaluation," in *Visual Models for Target Detection and Recognition*, E. Peli, Ed. Singapore: World Scientific, 1995, pp. 245–283.
[57] J.-B. Martens and L. Meesters, "Image dissimilarity," *Signal Processing*, vol. 70, pp. 155–176, Nov. 1998.
[58] VQEG. (2000, Mar.) Final Report From the Video Quality Experts Group on the Validation of Objective Models of Video Quality Assessment. [Online] Available: http://www.vqeg.org/

Zhou Wang (S'97–A'01–M'02) received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, in 1993, the M.S. degree from South China University of Technology, Guangzhou, China, in 1995, and the Ph.D. degree from The University of Texas at Austin in 2001.

He is currently a Research Associate at Howard Hughes Medical Institute and Laboratory for Computational Vision, New York University. Previously, he was a Research Engineer at AutoQuant Imaging, Inc., Watervliet, NY. From 1998 to 2001, he was a Research Assistant at the Laboratory for Image and Video Engineering, University of Texas at Austin. In the summers of 2000 and 2001, he was with Multimedia Technologies, IBM T. J. Watson Research Center, Yorktown Heights, NY. He worked as a Research Assistant in periods during 1996 to 1998 at the Department of Computer Science, City University of Hong Kong, China. His current research interests include digital image and video coding, processing and quality assessment, and computational vision.

Alan Conrad Bovik (S'81–M'81–SM'89–F'96) is currently the Cullen Trust for Higher Education Endowed Professor in the Department of Electrical and Computer Engineering, University of Texas at Austin, where he is the Director of the Laboratory for Image and Video Engineering (LIVE) in the Center for Perceptual Systems. During the Spring of 1992, he held a visiting position in the Division of Applied Sciences, Harvard University, Cambridge, MA. His current research interests include digital video, image processing, and computational aspects of biological visual perception. He has published nearly 400 technical articles in these areas and holds two U.S. patents. He is also the editor/author of the *Handbook of Image and Video Processing* (New York: Academic, 2000). He is a registered Professional Engineer in the State of Texas and is a frequent consultant to legal, industrial, and academic institutions.

Dr. Bovik was named Distinguished Lecturer of the IEEE Signal Processing Society in 2000, received the IEEE Signal Processing Society Meritorious Service Award in 1998, the IEEE Third Millennium Medal in 2000, the University of Texas Engineering Foundation Halliburton Award in 1991, and is a two-time Honorable Mention winner of the international Pattern Recognition Society Award for Outstanding Contribution (1988 and 1993). He was named a Dean's Fellow in the College of Engineering in the Year 2001. He has been involved in numerous professional society activities, including the Board of Governors for the IEEE Signal Processing Society (1996–1998), Editor-in-Chief of IEEE TRANSACTIONS ON IMAGE PROCESSING (1996–2002), member of the Editorial Board for THE PROCEEDINGS OF THE IEEE (1998-present), and Founding General Chairman for the 1st IEEE International Conference on Image Processing, held in Austin, TX, in 1994.

Hamid Rahim Sheikh (S'04) received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and the M.S. degree in engineering from the University of Texas at Austin in May 2001, where he is currently pursuing the Ph.D. degree.

His research interests include using natural scene statistical models and human visual system models for image and video quality assessment.

Eero P. Simoncelli (S'92–M'93–SM'04) received the B.S. degree in physics in 1984 from Harvard University, Cambridge, MA, a certificate of advanced study in mathematics in 1986 from Cambridge University, Cambridge, U.K., and the M.S. and Ph.D. degrees in 1988 and 1993, both in electrical engineering, from the Massachusetts Institute of Technology, Cambridge, MA.

He was an Assistant Professor in the Computer and Information Science Department at the University of Pennsylvania from 1993 to 1996. He moved to New York University in September 1996, where he is currently an Associate Professor in Neural Science and Mathematics. In August 2000, he became an Associate Investigator at the Howard Hughes Medical Institute, under their new program in computational biology. His research interests span a wide range of topics in the representation and analysis of visual images, in both machine and biological vision systems.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1261
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Next Generation Mobile Multimedia Communications: Media Codec and Media Transport Perspectives

Feng Wu, Guobin Shen, Kun Tan, Fan Yang, Shipeng Li
Microsoft Research Asia, Beijing, China
Email: {fengwu, jackysh, kuntan, fanyang, spli}@microsoft.com

ABSTRACT

In this article, we give an overview of several core technologies involved in the next generation mobile media communications system from both the media codecs and media transport perspectives. First, we introduce several video codecs that are suitable for mobile communications, including a low complexity and low bit rate codec for video conferencing and generic scalable video coding. We focus the discussion more on the state-of-the-art in scalable video coding and present an overview of the scalable extension of the H.264/MPEG-4 AVC standard. We then discuss several key topics regarding media transport, including error-resilient transport, congestion control, and service discovery for ad hoc and peer-to-peer (P2P) networks. We discuss error protection issues of wireless multimedia communications, streaming protocols of media communications and a service discovery scheme for ad hoc and P2P networks, even before mobile devices are connected. We point out some forward-looking new mobile media application scenarios that could be enabled by these core technologies, especially by scalable coding technology and the system built based on it. These include efficient wireless streaming, multiparty video conferencing and live streaming, collaborative streaming, incremental streaming, soft storage capacity, and intelligent synchronization between devices. We bring up several futuristic yet thought-provoking research directions that we think could make a substantial impact in mobile multimedia communications. They include *super-format* for ultimate media adaptation in wireless and mobile space and *distributed video coding* to move heavy computation loads from traditionally at the encoder to at the decoder.

Key words: next genearation multimedia communications, media codec, media transport

I. INTRODUCTION

In recent years, mobile multimedia communication services have grown rapidly due to the rich media content, broadband wireless networks, and flexible IP-based transport. First, besides traditional professional media content, more and more user-created content is now being distributed across the Internet. The Web 2.0 era is bringing even richer media content directly to the consumers. Second, new generations of wireless broadband networks, such as 3G, WiMax, and mesh networks are either being deployed or emerging. This greatly increases the bandwidth of wireless media communications and further facilitates the delivery and distribution of bandwidth-demanding multimedia applications such as video streaming and video conferences on mobile terminals. Furthermore, the transition to IP-based transport protocol brings even more efficient bandwidth usage and more flexibility to take advantages of software based solutions to optimize overall end-to-end system performance.

Multimedia communications can be classified into several different categories based on their specific application scenarios. From the contents perspective, they can be classified into *live content*, such as live event broadcasting, real time communication including voice over IP (VoIP), video conferencing, voice/video chat, and other methods in which multimedia content is generated on the fly; and *stored content* that is prepared ahead of time and stored in specific formats. From the delivery perspective, multimedia communications can be classified into *streaming* where media content is played back while bit streams are being delivered to the receiver; and *download* where media content is played back only after all the bits in the media streams have been delivered to the receiver. From the distribution model perspective, multimedia communications can be classified as a *server-client* model such as video on demand, where the media content is hosted-by or relayed through a central server; and

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1262
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

peer-to-peer (P2P) model, such as P2P streaming and P2P file sharing, where each peer can be both a content distributor and a content receiver and peers work together to obtain content. From the interaction perspective, media communications can be classified into *delay-tolerant* sessions in which delay is not a problem as long as the delay jitters are in a controlled range; and *interactive* sessions - in which real time and fast response to user interactions are a necessity. Please note that typical multimedia communication application generally can simultaneously belong to a few different categories. For example, a VoIP application belongs to categories such as *live* content, *streaming* delivery, a *server-client* model and *interactive* session. Similarly, a P2P media file download application belongs to category of *stored* contents, *download* delivery, *P2P* model and *delay-tolerant* session.

In mobile multimedia communication space, we find applications of almost every category of multimedia communications mentioned above. Despite the significant recent developments in mobile multimedia communications, mobile space continues to present unique challenges.

First of all, the bandwidth in mobile communication is always limited. Mobile media communications are mainly based on wireless communication channels to deliver data. Though recent development of next generation wireless networks, such as 3G, WiMax for Wide Area Networks (WAN) and Ultra-Wide Band (UWB) for Personal Area Networks (PAN), promises a significant boost in bandwidth, however, compared to wired networks, the bandwidth of wireless networks seem to be always a magnitude behind. Therefore, more efficient media codecs and transport protocols are always desired in mobile media communications.

Second, the nature of wireless channels is that they are error prone due to random errors, fading errors, and burst errors. Furthermore, the moving of a mobile terminal makes wireless channel conditions even more dynamic and unpredictable. How to devise an error-robust yet efficient mobile media communication system is another challenge. This would involve error-resilient media coding, error-resistant transport, and efficient error concealment post processing when errors do occur.

Third, wireless bandwidth and channel conditions vary significantly from time to time and mobile terminals come with different capabilities, such as screen resolutions, processing power, memory/storage, network access methods, and the like. To efficiently deliver media content to a mobile terminal would require the mobile media system to be able to adapt to the bandwidth, bandwidth changes, and terminal capabilities.

Furthermore, most of current multimedia communications in mobile space are of the *server-client* model in which a central server hosts the media content or relays the packets between mobile terminals. Can we leverage the power of P2P communication directly among clients to further enhance mobile media communication in a way similar to the wired P2P networks? There are several distinct challenges here:

1) Due to their mobile nature, mobile terminals are more unstable than fixed terminals with fixed networks attached.
2) Unlike always connected wired networks, wire-less networks need to first establish connections before they can communicate and exchange information. However, before connecting to a wireless terminal, a user should know which one has the information and services with which he would like to connect. Unfortunately this information is generally unavailable before the terminal is connected.
3) The mobile terminal normally needs to support both an infrastructure mode and P2P mode in order to take advantages of peerassisted media communications and services. This requires either hardware or software support in the mobile terminal.
4) Wireless network resources are even scarcer than wired resources and are normally more expensive than wired ones. The social implications of the wireless P2P networks and incentives to motivate people to participate and contribute to such networks present yet other challenges.
5) Last, but not the least, is the issue of battery life on mobile terminals. This is probably always a bottleneck for most mobile devices today. Any applications and services running on a mobile device should take into consideration underlying power constraints. That is, we should con-sider the complexity of such constraints when designing various processing algorithms and system architecture.

Every aspect of a mobile multimedia communication system itself can be addressed in a lengthy article. However, this is not our intention. In this article, we will use video communications as an example to cover just a few issues that we believe are central to a next generation mobile multimedia communication platform. These include video coding for mobile multimedia communications, multimedia transportation over wireless networks, and several new application scenarios and features that may enhance user experiences and help to promote mobile multimedia communication.

The rest of this article is organized as follows. Section 2 introduces a few novel and emerging video coding technologies that are suitable for mobile media communication. We provide a brief overview of low bit rate video coding for mobile communications and scalable video coding for generic video contents. In section 3, we will address transport issues related to media delivery over wireless networks, including error-robust transport and congestion control for mobile multimedia communications. We point

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1263
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

out some issues and solutions in wireless P2P and ad hoc networking such as resource and service discovery. In section 4, we propose several forward-looking application scenarios that would bring new mobile multimedia experiences to users. In section 5, we share our thoughts on some new research directions in mobile multimedia communications. Section 6 is our conclusion.

II. VIDEO CODING FOR MOBILE MULTIMEDIA COMMUNICATION

In mobile multimedia applications, video coding faces a threefold challenge:

1) Bandwidth is normally limited and requires more efficient video codecs.
2) Wireless access networks are normally heterogeneous and bandwidth and channel conditions generally are timevariant.
3) Capacity of mobile terminals also varies from one device to another.

A solution to the above challenges is to compress video into a scalable stream so that the bit stream can seamlessly adapt itself to any changing environment and device capacity. In this section, we will start with a low complexity low bit rate scalable codec [1] (called *Portrait* video) for mobile devices for live mobile communication scenarios and then give an overview of the start-of-the-art in general scalable video coding, including Fine Granularity Scalable (FGS) video coding, three-dimensional (3D) wavelet scalable video coding and MPEG-4 AVC/H.264 scalable extension.

2.1 Low complexity low bit rate scalable video codec - *Portrait* video

As we pointed out earlier, coding efficiency, scalability and low complexity are essential requirements for a video codec for mobile multimedia communications. In general, a generic scalable video codec is needed to efficiently handle general mobile video comm-unications. However, for some specific applications, some specially designed codecs can be even better. Portrait video codec [1] is such a special codec that aims to efficiently handle live mobile communication scenarios in which only head and shoulder scenes are involved in most cases.

Portrait video codec is such a special codec that it also takes human perception into account when coding live video sequences. When the channel bandwidth is very limited, it does not even encode video in full color. Rather, in the extreme case, it will convert video sequences into a series of black-and-white pictures with only two gray levels for each pixel: black and white. It can achieve as low as 9.6kbps video communication yet convey just enough video and audio information for the receiver to recognize who the person in the video is and what his facial expression is. As the channel bandwidth increases, the codec can be enhanced with four-level video, 16-color video and up to full color video. Such a video codec provides both rate scalability and complexity scalability according to the conditions in a live communication session. The different codecs can be switch back and forth on the fly to adapt to the channel condition changes and mobile terminal changes. Figure 1 shows the generation process of multi-level video using a four-level video as an example. Figure 2 gives the bitstream structure of the multi-level Portrait video coding. We can see that each higher level video contains all the bit stream of a lower level video. Each bit stream of a higher bit plane details the gray levels represented by a bit stream in its lower bit plane. This provides a natural embedded bitstream. Portrait video software can be downloaded from http://research.

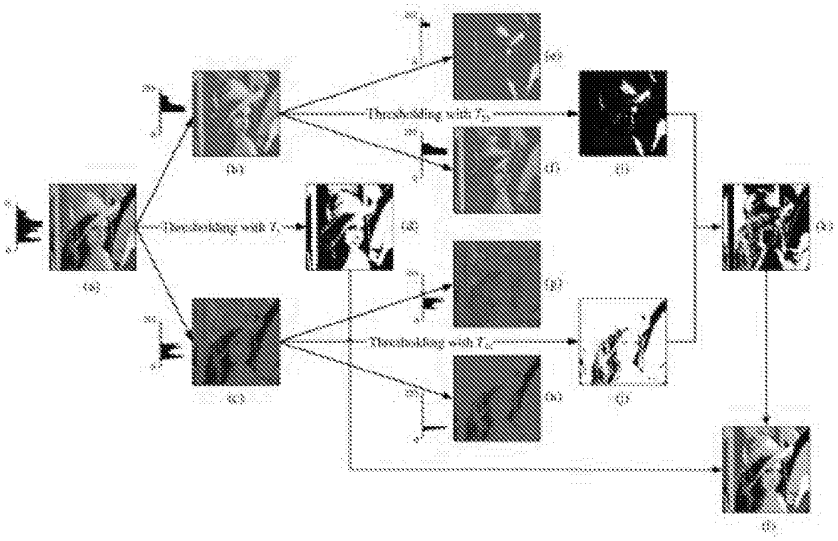


Fig.1 Generation of four-level video in Portrait Video

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1264
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

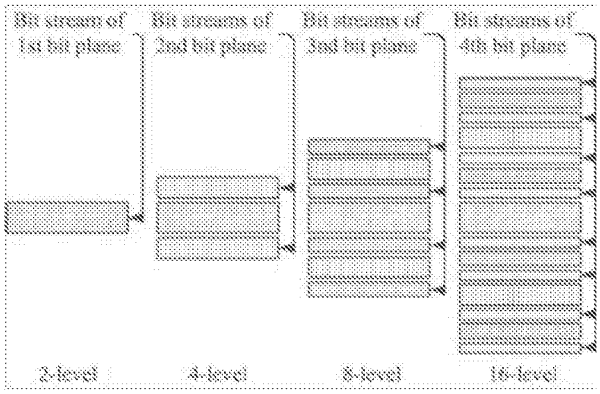


Fig.2 Bitstream structure of multi-level video coding in Portrait video

microsoft.com/mcom/portrait/.

2.2 Generic scalable video codecs

Although portrait video can address the bit rate and complexity scalability issue of live video communication that is involved in head and shoulder scenes, it cannot solve the scalability problem for generic video, for example, those pre-stored compressed contents whose bit rate and quality difficult to adjust on the fly. Furthermore, it does not provide effective temporal scalability and spatial scalability.

A generic scalable codec should include quality (SNR) scalability, temporal scalability, and spatial scalability. Scalable coding refers to coding of an image or a video sequence into a bitstream that is partially decodable to produce a reconstructed image or video sequence at a possibly lower spatial, temporal, or quantization resolution. Figure 3 shows the relationship between different scalabilities for a video bitstream. The principal concept in scalability is that an improvement in source reproduction can be achieved by sending only an incremental increase in rate over the current transmitted rate that is achieving a coarser reproduction of the source. SNR, spatial, and temporal scalability are all important in applications. A key feature of scalable coding is that the encoding only needs to be done once and a single scalable bitstream will fit in all applications with different target bit-rates, or spatial or temporal resolutions. Scalable coding can bring many benefits in practical applications: potential high coding efficiency through rate-distortion optimization (e.g. JPEG-2000); easy adaptation to heterogeneous channels and user device capabilities and support for multicast; better received picture quality over dynamic changing networks through channel adaptation; robust delivery over error-prone, packet-loss-ridden through unequal error protection (UEP) and priority streaming since some enhancement bitstream can be lost or discarded without irreparable harm to the overall representation; reduced storage space required to store multiple bitstreams otherwise; no complicated transcoding needed for different bit rates and resolutions.

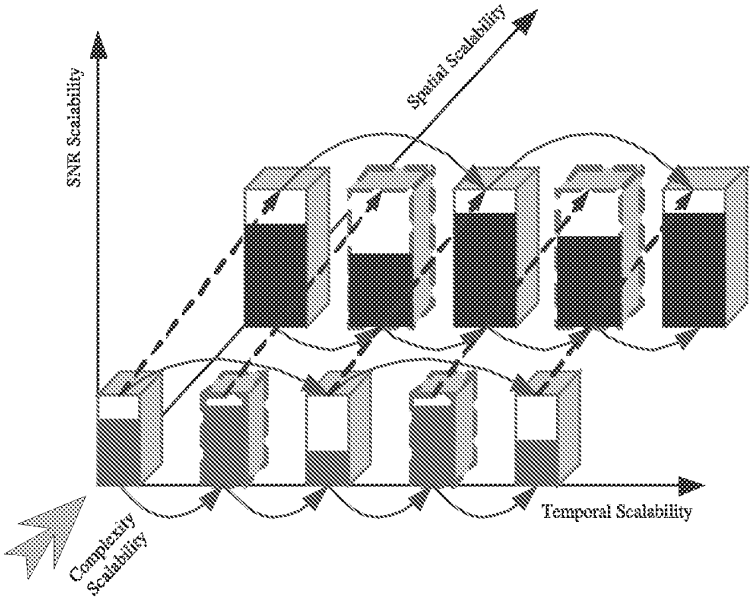


Fig.3 Relationship between different scalabilities

In the following subsections, we will briefly introduce the state-of-the-art scalable video coding scheme to give readers a high-level idea of the status of scalable video coding research in the world.

2.2.1. FGS scalable video coding

Among the SNR scalable video coding techniques, MPEG-4 Fine Granularity Scalable (FGS) coding has become prominent due to its fine-grain scalability [2][3]. In MPEG-4 FGS, a video sequence is represented by two layers of streams with identical spatial resolutions. These are referred to as the *base layer stream* and the *enhancement layer stream* (see Figure 4). The base layer stream is coded with non-scalable coding techniques, whereas the enhancement layer stream is generated by coding the difference between the original DCT coefficients and the reconstructed base layer coefficients using a bit-plane

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1265
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

coding technique. The residual signal is represented with bit planes in DCT domain, where the number of bit planes is not fixed, but based on the number of bit planes needed to represent the residual magnitude in binary format.

One major feature of MEPG-4 FGS is that the base layer and all bit planes in a predictive picture are always predicted from the reconstructed version of the base layer in the reference picture. Any losses or corruptions of one or more bit planes during transmission have no effects on the pictures that follow. Therefore, MPEG-4 FGS provides excellent error recovery from occasional data losses or errors at the enhancement layer. However, since the prediction is always based on the lowest quality base layer, MEPG-4 FGS also suffers from severe degradation on coding efficiency.

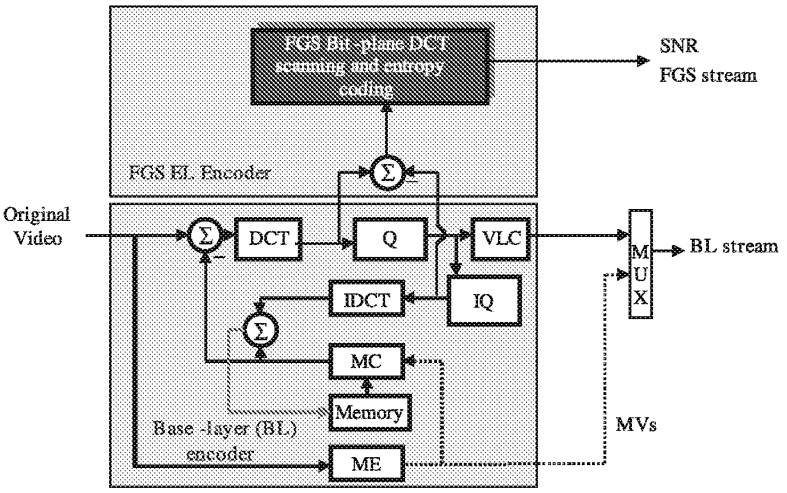


Fig.4 MPEG-4 FGS encoder

How can one introduce the temporal prediction into the FGS enhancement layer coding without severe drift errors? Several alternative techniques have been proposed to address this problem.

Progressive FGS (PFGS) proposed in [4][5] explores a separate motion compensation loop for the FGS enhancement layer to improve compression performance and provides means to eliminate the drift error. There are two key points in PFGS coding. One is to use as many predictions from the enhancement reference layers as possible (for coding efficiency), instead of always using the base layer as in MPEG-4 FGS. The other point is to keep a prediction path from the base layer to the highest quality layer across several frames to assist in error recovery and channel adaptation. It enables the lost or erroneous higher quality enhancement layers to be automatically and gradually reconstructed from lower layers over a few frames. Thus, PFGS trades off coding efficiency for drift error reduction.

In [6], a robust FGS (RFGS) technique was presented by incorporating the ideas of leaky [7] and partial predictions to deal with the drift error. In RFGS, the high-quality reference used in the enhancement layer compensation loop is constructed by combining the reconstructed base layer image and part of the enhancement layer. A frame-based fading mechanism is introduced to cope with the mismatch error. Specifically, at each frame, a uniformly leaky factor between 0 and 1 is applied to the enhancement layer before adding to the base layer image to alleviate error propagation.

Furthermore, several techniques are proposed to achieve more flexible trade-off between drift errors and coding efficiency at the macroblock level rather than at frame level. The macroblock-based PFGS (MPFGS) is one such scheme [8][9]. In MPFGS, three INTER modes, HPHR, LPLR and HPLR, are proposed for the enhancement layer macroblock encoding. In fact, the HPHR mode is used to get high coding efficiency by using higher quality reference, while the HPLR mode is imposed to attenuate drifting errors by introducing the mismatch errors into the encoding process. Assuming the base layer is always available at the decoder, LPLR and HPLR modes will help to reset the drift errors potentially caused by HPHR mode. A decision-making mechanism is presented in MPFGS to choose the optimal prediction mode for each enhancement layer macroblock by considering the error propagation effects and taking advantage of the HPLR mode to achieve flexible trade-off between high coding efficiency and low drifting error.

2.2.2. 3D wavelet scalable video coding

3D wavelet video coding schemes do not use traditional motion compensation. Instead, correlations across pictures are exploited by temporal transform. The idea of temporal extensions of subband decompositions appeared in the late 80s, with the works of Karlsson and Vetterli [10] and Kronader [11]. In these works the classical temporal closed-loop prediction scheme was replaced by a temporal subband decomposition, which did not take into account any motion compensation. However, it was shown that motion prediction was also important in these schemes [12][13] in order to reduce the detail energy subbands, thus leading to much better compression performance and visual quality, and ideally the temporal transform should be applied along the motion trajectories.

The simplest temporal wavelet transform is the Haar one, performing sums and differences of pairs of frames to obtain

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1266
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

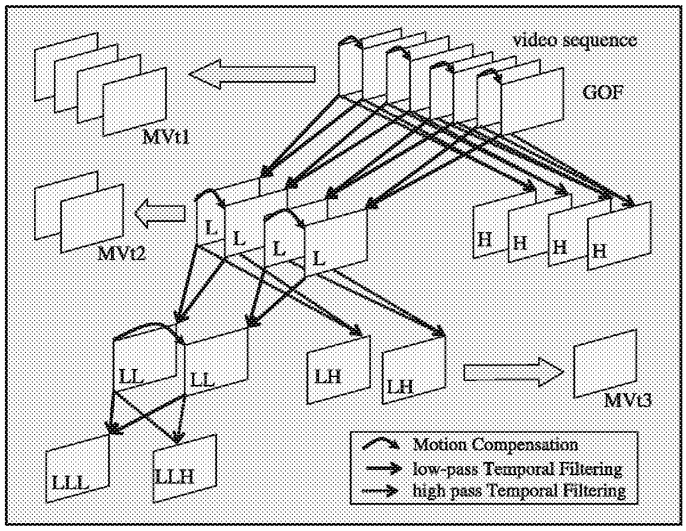


Fig. 5 Temporal Haar wavelet decomposition of a GOF

the approximation, respectively the detail subbands. It is illustrated in Figure 5 on a group of frames (GOF) of eight frames, which allows performing a maximum of three levels of dyadic decomposition. Due to the two-tap, low-pass and high-pass filters and down-sampling of factor 2, no boundary problems appear when decomposition a GOF of size 2^L into a number of up to L resolution levels.

However, when we introduce *motion-compensated temporal filtering* (MCTF) into the coding framework, motion invertibility becomes an important problem. For example, in block-based motion compensated prediction, which is the most usual technique for temporal prediction, the same area in the reference frame can be used to predict several areas in the current frame, while parts of the reference frame are not used at all for prediction. This gives raise to multiple connected and unconnected pixels. To effectively deal with the problem, [14] proposes a new and general lifting structure that unifies all the previous work to solve this problem and enables any traditional motion compensation techniques in block-based motion prediction coding to be easily adopted in the MCTF frameworks. The core of this work is a so-called *Barbell lifting* scheme, in which instead of a single pixel value, a function of a set of pixel values is used as the input to the lifting step. The Barbell lifting scheme essentially moves any existing effective motion prediction schemes in traditional video coding to the MCTF frames.

After 3D (temporal and spatial) wavelet analysis, a video sequence will be decomposed into a certain number of 3D subbands. For example, a three-level wavelet (motion compensated) decomposition is performed in the temporal direction, followed by a three-level 2D spatial dyadic decomposition within each of the resulting temporal bands. The next step in 3D wavelet video coding is to encode the transformed 3D wavelet coefficients in each subband efficiently. Since the subband structure in 3D wavelet decomposition for video sequence is similar to the subband structure in 2D wavelet decomposition for image, it is natural to extend many existing 2D wavelet-based image coding techniques, such as SPIHT [15], EBCOT [16] and EZBC [17], to the 3D case. As a matter of fact, almost all the existing 3D wavelet coefficients coding schemes are using one form of these 3D extensions, such as 3D SPIHT [20], 3D ESCOT [18][19] and 3D EZBC [20][21].

2.2.3 MPEG-4 AVC/H.264 scalable extension

MPEG-4 AVC/H.264 scalable extension enables scalabilities while maintaining the compatibility of the

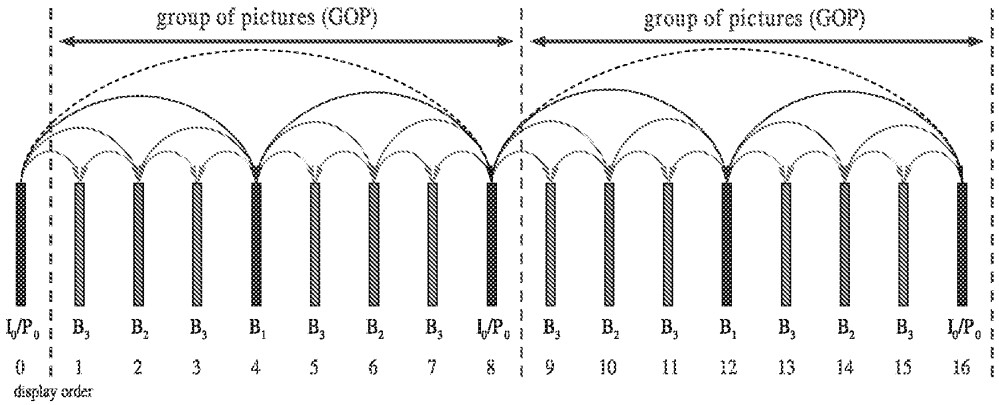


Fig.6 4-level hierarchical-B prediction structure

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1267
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

base layer to MPEG-4 AVC/H.264 standard. MPEG-4 AVC/H.264 scalable extension provides temporal, spatial, and quality scalabilities. Those scalabilities can be applied simultaneously. In MPEG-4 AVC/H.264, any frame can be marked as reference frame to be used for motion prediction for the following frames. Such flexibility enables various motion compensated prediction structure.

The common prediction structure used in MPEG-4 AVC/H.264 scalable extension is a hierarchical-B structure, as in Figure 6. Frames are categorized into different levels. And B-frames of level i use neighboring frames at level i-1 as references. Except for update steps, MCTF and hierarchical-B have the same prediction structure. Actually at the decoder, the decoding process of hierarchical-B and that of MCTF without update steps are the same. Such hierarchical prediction structure exploits both short-term and long-term temporal correlation as in MCTF. The other advantage is that such structure can inherently provide multiple levels of temporal scalability.

To achieve SNR scalability, enhancement layers, which have the same motion compensated prediction structure as the base layer, are generated with finer quantization step sizes. At each enhancement layer, the differential signals to the previous layer are coded.

To achieve spatial scalability, the lower resolution signals and the higher resolution signals are coded into different layers. The coding of the higher resolution signals uses the bits for the lower resolution for prediction. MPEG-4 AVC/H.264 scalable extension sets a constraint on the interlayer prediction among different resolutions in which only intra-coded macroblocks are reconstructed to predict the higher resolution, while for inter-coded macroblock, only the motion compensated residue signals are allowed to predict the corresponding residue signals at the higher resolution. The advantage of such a constraint reduces the decoding complexity because the decoder does not need to do motion compensation for the lower layer. The drawback is that such constraints may bring a coding performance penalty.

Scalable coding of multimedia contents provides some useful properties for mobile multimedia communications. It brings network bandwidth and channel adaptation, device adaptation, and error-resilience features. It provides power constrained media processing since the mobile terminals now can choose which scalability layer to process according to their respective power levels. The drawbacks of scalable video coding are that it generally also brings coding efficiency loss to some extent and the decoders are normally more complex than single layer video codecs. However, these various advantages of scalable coding cannot be fully enabled without a corresponding scalable transport framework.

III. MULTIMEDIA TRANSPORTATION OVER WIRELESS NETWORKS

The unique characteristics of mobile wireless networks pose many challenges for multimedia applications. Wireless links generally have higher packet loss and bit error rate. The bandwidth of wireless links is usually limited and also fluctuates with changing of the environment. This is especially true when mobile devices roaming from one network to another network (e.g. from GPRS network to WLAN or vice versa), when network conditions may change dramatically. Traditional transport protocol, i.e. TCP, does not work well to deliver multimedia contents and therefore there raises new demands for new transport protocols for multimedia applications. IETF has developed a framework that contains several standards for delivery of multimedia data, such as a real-time transport control protocol (RTP/RTCP), real-time streaming protocol (RTSP), session initiation protocol (SIP), and session description protocol (SDP). However, many aspects in the protocols still require research effort. In the following subsection, we provide an overview of some recent research in transport error protection, congestion control for multimedia applications and service discovery.

3.1 Transport error protection

Besides the packet loss due to buffer overflow in the intermediate routers, packet may also be lost because of errors in the wireless channels. Thus, efficient error protection schemes are essential for improving end-to-end media quality. Traditional transport protocols use retransmission to recover packet losses. That is, when a packet is detected to be lost from the sender (either from an acknowledgement or a timeout), the sender will resend the entire packet. This, however, is not suitable for applications with strict delay constraints, such as voice communications. Therefore, forward error correction (FEC) is generally used as the error protection approaches in real-time multimedia communication. FEC is a channel coding technique that protects the source data at the expense of adding redundant data during transmission [22]. In the case of media transmission, where delay requirements are not that strict (e.g., delay-tolerant session, such as video/audio delivery over a single wireless channel), retransmission may still be applicable and play a complementary role to FEC. In [23], a hybrid FEC/ARQ method was adopted and the delay bound was achieved by limiting the number of retransmissions.

FEC requires redundancy to protect from errors, therefore adding transmission overhead. Given the limited network bandwidth, how to allocate these protection-bit is an interesting research problem. One idea is to allocate the protection-bits unequally to different contents in multimedia application. For example, in layered scalable media

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1268
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

codec, multimedia data are usually divided into a base layer and multiple enhancement layers. In addition, the correct decoding of enhancement layers depends on the error-free delivery of the base layer. Thus, the base layer is more important, and stronger FEC protection should be applied to the base layer data than the higher layer parts [24]. Since the wireless channel may dynamically change it is also a natural idea to adjust the FEC protection level in response to the underlying changing network conditions. For example, GSM systems can dynamically distribute voice data and channel coding among the overall bandwidth to the possible best voice quality. In [25], the authors study how the protection level of every layer in video bit-stream is dynamically adjusted according to the changing network conditions.

A more generic form of aforementioned problem is called bit allocation problem, which jointly considers both source coding and channel coding. A generic analytic formulation of the bit-allocation problem is as follows

$$\min D_T(D_S, D_C) \quad s.t. \quad R_S + R_C \le R_T \tag{1}$$

where R_T is the total available bandwidth, R_S and R_C are the rates for the source coding and channel coding, respectively and D_T is the expected end-to-end distortion. Smaller distortion means higher media quality. In general, D_T consists of two components, source distortion D_S and channel distortion D_C. Source distortion is caused during the media source encoding. Variable encoding methods such as quantization, motion estimation in video coding, linear prediction in voice coding, and rate control can be the cause of source distortion. Channel distortion occurs when fragments of media stream are lost due to network congestion, or incorrectly received due to wireless channel noise.

From the above formulation, we can see that both source coding and channel coding parameters can affect final media quality. Joint Source-Channel Coding (JSCC) schemes are thus proposed to achieve the optimal end-to-end quality by adjusting the source and channel coding parameters, simultaneously. From the source-coding point of view, adjusting quantization parameters or the entropy coder to control source rate [26], selecting inter- or intra-coding mode to trade off the coding efficiency and the error resilient ability [27], are key issues that can be jointly considered with channel coding. In [28], an integrated JSCC scheme has been proposed to study the performance of FEC and ARQ. Meanwhile, the joint effect of FEC/ARQ and error-resilient source coding is considered.

3.2 Congestion control for multimedia

One important issue for delivering multimedia content over wireless networks is congestion control. Congestion control algorithms are used to prevent the congestion collapse in networks and therefore are important to keep the network stable. Transport Control Protocol (TCP) employs an AIMD congest control algorithm that increases the window by one packet when no loss is detected in a round and cuts the window in half when a loss is detected [30]. However, this congestion control algorithm has two major problems when applied to multimedia communications. First, the TCP congestion control algorithm introduces a large variance in the sending rate, which adversarial to multimedia delivery. Second, TCP relies on packet loss as the only indication of congestion. However, in wireless networks, congestion may not be the only source of packet losses, and wireless link errors may also cause packet losses. As a consequence, the TCP congestion control algorithm may not be able to efficiently utilize the wireless channel.

There has been an active research field to design congestion control for multimedia communication. One of most important constraints for these new algorithms is TCP-friendliness, which is roughly defined as the long-term throughput of a multimedia stream not exceeding a TCP flow in the same situation [29].

In general, there are two sorts of TCP-friendly streaming control protocol: window-based and equation-based. The window-based congestion control approaches remains a congestion window as in TCP. The adjustment of the congestion window is tuned to have the same long-term throughput as TCP, but with less fluctuation in sending rate. For example, the rate-adaptive protocol (RAP) [31] mimics the behavior of the TCP congestion window and an acknowledgement is triggered by every incoming packet on the receiver side to measure packet loss and RTT. Equation-based congestion control algorithms adjust the sending rate based on the throughput equation of TCP [32]. They measure current network parameters to feed into the model to determine the proper sending rate. TFRC is a well-known streaming protocol based upon TCP throughput equation [33].

However, while the aforementioned the streaming protocols work well in the wired Internet case, they have the same deficiency as TCP in wireless networks, in which packets may be lost because of channel errors. Random wireless errors may wrongly trigger congestion control behavior and, as a result, throughput is degraded.

Many schemes have been proposed to improve the performance of transport over wireless networks by separating wireless links from the Internet. These schemes are called *split-connection*. A proxy or agent is deployed in the edge of the wired and wireless part of the network [34][35][36]. Different congestion control schemes are used in wired and wireless part. In the wired part, the aforementioned TCP-friendly streaming protocol can be used; while in the wireless part, a special protocol is designed to compensates for the unreliability and unpredictability of the wireless link. However, the split-connection approaches require the deployment of the proxies and increase the cost for network operators.

Many other approaches are proposed based on end-to-end

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1269
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

information. They develop several heuristics to differentiate the congestion packet losses from the random erroneous losses. In [37], Biaz and Vaidya used packet inter-arrival time to differentiate the cause of losses. And in [38], Cen et al. extended their idea and further incorporated relative one-way trip time (ROTT) as an additional metrics to discriminate the losses. In [39], the authors used packet pair measurements to build a hidden Markov model to achieve loss differentiation. Tan et al. in [40] proposed using the burst length of the packet loss in a round-trip time as a heuristic for differentiating the erroneous losses. And TCP-Veno proposed a packet loss differentiation method with the help of packet delay information. It claims that only packet losses during the period when RTT varies greatly are congestive losses [41]. Another way to handle packet loss ambiguity is to directly estimate the available bandwidth of the network path. TCP-Westwood mines the ACK streams to derive the "Eligible Rate," with which the sender sets the congestion window after detecting a loss event [42] instead of halving it.

3.3 Service discovery for ad hoc or P2P wireless networks

As we mentioned in the introduction session, in ad hoc or P2P wireless networks, we need first to connect to a mobile device before we know what kind of services it can provide. However, we need to know what kind of services each mobile device can provide before we know which one to connect.

With the proliferation of IEEE 802.11 enabled devices, there are increasing demands for spontaneous collaborative services over temporarily formed ad hoc networks, especially when there is a lack of network infrastructure support. However, the current 802.11 devices are mostly used in a "Network centric" way, in which a device must be connected to a network before it can discover services or send out requests. This approach is not suitable for spontaneous collaborative services because network setup should take place only after a certain service is requested.

To this end, a novel approach, named EZSetup [43], is proposed. It utilizes the IEEE 802.11 ad-hoc mode to support collaborative service while fully complying with the current standard. EZSetup encodes the device and service information in the periodical beacons and leverages them as signaling mechanisms. Devices can then discover one another and learn service requests and invitations prior to forming a network. Preliminary evaluation has demonstrated that EZSetup is an effective mechanism and convenient tool for spontaneous collaborations among 802.11 enabled devices.

As mentioned, a new mechanism is needed to assist "service centric" collaboration over spontaneous networks. This new mechanism should satisfy the following three requirements:

1. Be able to discover neighboring WiFi devices and their supported services;
2. Be able to convey requests for collaborative services to peers. In other words, the initiator of the collaboration should be able to *invite* other participants to automatically form a spontaneous network and start the same service;
3. Do not rely on any preinstalled network infrastructure. This implies that the devices are *disconnected* in the first place and no IPbased packets can be exchanged.

Existing service discovery and invocation mechanisms, like SLP [45] and UPnP [46], heavily depend on underlying communication infrastructure, e.g. TCP/IP conn-ections, to transmit control messages among networked devices, and therefore are not applicable for this spontaneous networking scenario.

Indeed, the IEEE 802.11 standard has defined a way to discover an existing network, via periodic beacons from APs or the stations in *ad hoc* mode. A beacon is a specific IEEE 802.11 packet that announces the existence of a network, identified by a 32-byte *Service Set Identification*, or SSID [43]. A mobile device can scan the channels across a range of the radio spectrum and receive these beacons. In current IEEE 802.11 standard, a beacon contains little information that is only enough for a device to *connect* to a network, and lacks of

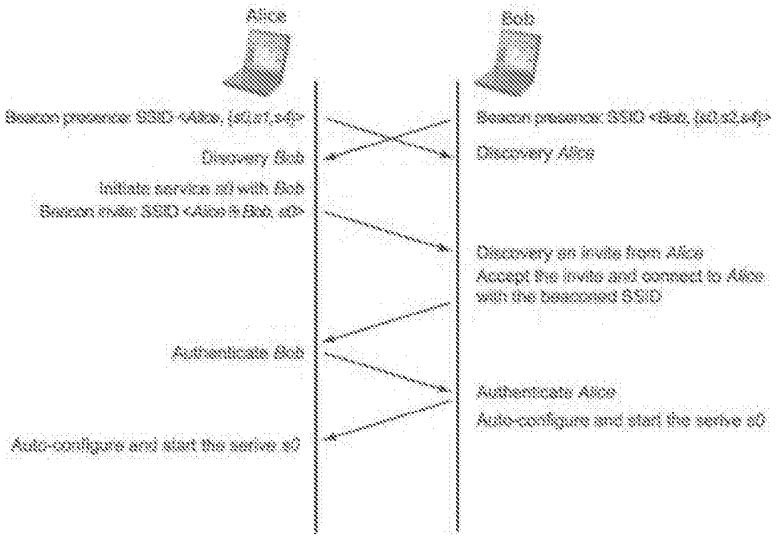


Fig.7 Procedures to initiate a collaborative service between user Alice and Bob using EZSetup

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1270
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

important semantics to support for spontaneous collaborative services: 1) application service announcement and discovery; and 2) the request/response paradigm for invoking a service.

However, these semantics can be supported without violating the IEEE 802.11 standard. Specifically, we introduce a new format for the SSID field in the beacon packet. Our scheme can encode rather rich information within limited spaces (i.e. 32 bytes), and is very flexible to support a number of common collaborative services over spontaneous networks.

The common procedures to initiate a spontaneous collaborative service using EZSetup are outlined in Figure 7. When disconnected, devices are configured themselves in ad hoc mode to broadcast a *presence beacon* on a randomly selected channel, which encodes the device name in their SSIDs as well as a set of collaborative services that are available on the devices. By scanning over all possible channels, a device can learn all devices nearby that are running EZSetup and the services they are carrying. Note that with scan function, a device can also find other conventional WiFi networks, but SSIDs generated by EZSetup are self-validated and can be easily separated from conventional SSIDs (see details in Section 3.1 *i*).

When *Alice* wants to initiate a collaborative service, say *s0*, with *Bob*, EZSetup will reconfigure an *invitation beacon*, which encodes the participants' information (i.e. *Alice* and *Bob* in this example) and the expected service (see details in Section 3.1 *ii*). Still with scanning function, *Bob* can detect this encoded beacon and determine that he is one of the invitees from *Alice* to perform s0. Under *Bob*'s permission, EZSetup indicates the underlying Network Interface Card (NIC) to connect to *Alice* device using the information contained in the beacon. Then, the standard actions to form an IEEE 802.11 ad hoc network take place. After mutual authentication, the devices self-configure their IP addresses (see details in Section 3.2), and the corresponding service is automatically launched. Up to now, a spontaneous network is formed, and *Alice* and *Bob* can start service *s0* over this network.

Once the mobile terminal can discover each other's services can capacity, a wireless P2P networks can be easily formed to enable a plural of applications similar to the wired counterparts.

IV. NEW SCENARIOS FOR MOBILE MULTIMEDIA COMMUNICATIONS

In previous sections, we have discussed some advanced coding and networking technologies that will be the pushing power of the next generation multimedia experience. In this section, we will outlook some new scenarios and features that we have envisioned. We start by examining what new scenarios can be enabled by the scalable coding technology.

Scalable media in general enables many new applications and significantly improves many existing ones. There is usually a misconception that scalable media can only bring benefits to the dynamic changing environment such as Internet and wireless streaming. However, the applications of scalable media are way beyond our original objectives. Examples given below are just some snapshots of the potential applications we are considering. We will use the SMART system[47] as an exemplar system but the applications are generally applicable to other scalable systems. The SMART system is an end-to-end scalable media framework that includes scalable codecs and scalable transport. For details of SMART system, please refer to[47]. Note that from the beginning to the end, the server only needs a single scalable bitstream for one particular media content to provide all these different services. The production and maintenance cost is thus greatly reduced.

4.1 Internet streaming

With the SMART Video codecs and the SMART system, SMART technology can provide the following benefits.

1) SMART technology enables *fast start and fast random access* without requiring burst bandwidth. The server can first use the normal bandwidth to turbo stream the low bit rate base layer to the client and quickly build up enough base layer content in the client buffer for the playback to start. The lowquality base layer will be gradually and gracefully enhanced to full quality as more enhancement contents are delivered afterwards. This feature is essential for quick channel changes and quick media browsing.
2) SMART technology enables *effective handling of network dynamics*. When the network bandwidth drops due to congestion, for example, the SMART system can detect the change and simply reduce the bit rate of the scalable video bitstream to match the available channel bandwidth which in turn alleviates the congestion. When the network bandwidth increases, the SMART system can detect the change and increase the video bit rate to achieve better video quality that is maximally allowed by the available bandwidth. Note that the Multiple Bit Rate (MBR) approach used in current streaming systems can also provide some bandwidth adaptation by switching among several different nonscalable bitstreams with different bit rates. However, it can only provide coarse adaptation and lagged switching.
3) The most significant benefit of SMART technology in streaming applications is that it can significantly reduce the chance of *freezing and rebuffering* phenomenon that commonly exists in any streaming system using nonscalable bitstreams. Even with the MBR approach and intelligent bandwidth detection,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1271
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

existing stream-ing systems still suffer from rebuffering glitches due to lagged switching. This is because they must flush the current buffer and wait for the next key frame in the new bitstream to arrive. On the other hand, the SMART system can use unequal error protection and selective ARQ techniques to protect the more important data, (e.g., the base layer) from packet losses and errors, and then can lose the less important data (e.g., the enhancement layers). The base layer data can be made available at the client side with high probability, thus resulting in a smooth playback experience.

4.2 Multiparty video conferencing and live streaming

Currently, when starting a multiparty video conference with participants behind heterogeneous networks with different bandwidth, we are forced to encode the video at a bit rate not exceeding the minimal bandwidth of all the connections. In other words, the video quality is limited by the minimal bandwidth. However, with SMART technology and multicast technology (either routersupported, IP-multicast or application level multicast), the video can be encoded into a scalable bitstream and each participant may choose to receive the portion of bitstream that best meets its own bandwidth requirement and device capability. Obviously, all these technologies can be also applied to live streaming events such as sports, concerts, and the like.

4.3 Wireless streaming/roaming

Wireless streaming is another important application of SMART technology. With SMART technology, wireless streaming shares the same set of benefits as the Internet streaming in cases of packet losses and bandwidth changes. However, the data packets in wireless networks are more error prone as compared to the wired counterparts. Though these bit errors are generally encapsulated in data packets and appear to be packet losses to users, there are certain wireless network protocols such as UDP Lite that can pass the errors to the decoder. SMART technology can tolerate these errors at the coding layer and limit the impact of such errors to within a few frames, thanks to the strong drifting prevention technique and excellent error resilience capability in SMART codecs.

Another application of SMART technology is wireless roaming, including network roaming and device roaming. As we know, there are a variety of different wireless networks available today, for example, wireless WAN (GPRS, CDMA1X), and wireless LAN and the emerging UWB. More and more portable devices equipped with two or three kinds of radios for different networks. Seamless roaming between these different networks is highly desired to ensure a pleasant streaming experience. SMART technology enables such seamless roaming because it never needs to switch between different bitstreams. All it has to do is to send a higher bit rate when using wireless LAN and send less bit rate when using wireless WAN. The same video buffer and its buffered contents can be shared during roaming and it never needs to be flushed.

4.4 Collaborative streaming

Due to the intrinsic fine granularity scalability of the bitstreams generated by the SMART coding technology, the video quality can be enhanced collaboratively in the follow ways:

- In a P2P networking scenario, peers can negotiate and determine to streaming different portions of the same bitstream and exchange the received bitstreams so as to aggregate more effective bandwidth (which implies better video quality)[48];
- In a wireless streaming setting, two users can use their GPRS or CDMA1x connections to stream a different portion of the bitstream and exchange the received bitstreams through other wireless networking like BlueTooth or Wi-Fi.

Similarly, the connection between clients can be used to combat the packet losses and/or the errors in the bitstreams as well[49].

4.5 Incremental streaming

A scalable bitstream is additive by definition. Therefore, when a cache is enabled, the more times the content is streamed, the better the quality results. That is, the video quality can be improved consistently with the number of viewing sessions. For example, the video clip is very interesting and a user may watch it many times. Each time he watches, the quality will be improved. This is especially useful for a mobile device where the network bandwidth is limited. In another example, in a corporate or campus network, a proxy is usually deployed at the gateway. For some hot news clip, many users in the organization may watch the same clip. In this case, the more users watch the clip, the higher the quality that will result.

4.6 Home A/V distribution

Home is where most people spend most of their time to learn, entertain, play, and communicate with one another. With the rapid penetration of home networking, portable, and fixed media devices, efficient and smooth home A/V distribution remains a challenging problem. Similar to r Internet streaming, SMART technology can enable glitch-free media distribution over data networks, both wired and wireless. It also allows media content to be encoded only once to serve all different devices. Examples include an Xbox, a pocket PC, a Portable Media device, a Media Center PC, or the like. This is especially important for

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1272
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

home media servers since there processing power is limited and frequent transcoding between different formats is not realistic. If SMART format becomes the primary media format at home, all the problems with home A/V streaming can be easily solved.

An interesting idea to use SMART technology is turning a portable device with Wi-Fi connection into a remote control with media preview, i.e., a so-called picture-out-picture (POP) capability. While the Media Center PC (that connects to a large display like TV) is playing back one streaming session (say Channel A), the pocket PC user can use it to monitor another session (say Channel B). The user can switch the two channels freely when there is a necessity such as some exciting shots appears in Channel B and the user wants to share with others watching Channel A on the Media Center PC.

4.7 Soft capacity media storage

When recording video data into a Digital Video Recorder (DVR), there is always a chance that the hard disk or memory may run out of space. In that instance, one would face the dilemma between either to delete some old recorded shows or to stop recording new shows. With SMART video coding technology, a more preferable third choice is available: just gracefully degrade the quality of the recorded shows by optimally throwing away some enhancement bits and make room to record the new show. This feature can also apply to mobile/portable media player devices so that the users have total control of quality/capacity tradeoff without re-encoding/transcoding.

4.8 SMART sync

This feature is actually a combination of fast adaptation capability, incremental streaming and soft storage features. In normal sync scenario, when people sync files, they have to sync files one by one at full quality. On the other hand, for scalable media, a bunch of files can be synchronized simultaneously but at a reduced quality. In this way, people can get all the content quickly. At a later time, the user can retrieve higher quality content if there is more time available for the media sync or more bandwidth available when consuming the content. Also, in case of not enough storage for new syncs, with scalable media, people have the option of discarding some of the enhancement layers (which usually sacrifices little quality but saves much more space) so that more files can be accommodated without deleting other already existing files,

4.9 Video processing and content analysis

Many video processing and content analysis algorithms are actually not applied directly to a fully decoded video sequence for complexity considerations. Instead, they are often operated on the thumbnail or a much lower resolution sequence. SMART technology allows users to rapidly process and analyze the video content based on a video sequence generated from partial decoding of a scalable bitstream, i.e., only decode the base layer.

V. RESEARCH DIRECTIONS

In this section, we present some research directions that worth pursuing. We mainly raise questions instead of providing answers.

5.1 Super-format

The universal access requirement to multimedia content is ever increasing along with the more diversified access manners and heterogeneous terminal devices. As a result, there is an urgent requirement for transcoders to convert video contents from one format to another, together with parameter changes, to better serve different purposes[50][51]. Research on scalable video coding has been active in recent years. However, the transcoding business will long exist since scalable coding only fulfills the requirement of coding parameter adaptation, but not for other requirement such as format change and specific functionalities such as logo insertion, VCR-like functionality for streaming, and others. There is also on-going work to transcode non-scalable bitstreams to scalable ones[52] and that performs complex scalable transcoding[53].

Ironically, while the new coding technologies such as H.264 and H.265 emerge, the content world is dominated by few formats like MPEG-2 (i.e., DVD), MPEG-4, Windows Media Video, Real Video, QuickTime, DiVX, and others. It turns out that different formats are indeed a hindering factor for the widespread use of multimedia content and it is this real situation that drives the research of transcoding. However, in our opinion, the transcoder is only a rescue, not a driving force. In fact, transcoding is not a cheap rescue. Generations of transcoding imply consistent quality degradation. The number of transcoders grows exponentially since a transcoder is needed between any two video formats even if we exclude other desired functionalities. Therefore, we should strike out to find a technology for different coding frameworks can be synergized. Based on this, we have come up with the *Super-Format* concept.

By Super-Format, we mean that a bitstream should be fully self-contained. The bitstream contains two portions: the control part and the data part. The control part is descriptive and contains all the necessary information about how to configure and change a decoder on demand and on the fly so as to correctly handle the subsequent data part. The control part should be well organized and based on extensible markup language such as XML. The data part is the actual bits of content that can be

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1273
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

generated and organized to obey the rules identified in the control part. The control part and the data part can be intertwined.

Besides annulling the necessity of pre-installing many decoders and transcoders, Super-Format also enables easy editing of bitstreams, thanks to the capability of generating decoders on the fly. It will be simple to compose a new bitstream from existing ones by simple cut-and-paste the bitstreams in the compressed domain, without a need decompress the bitstream. New functionalities like down-sampling towards smaller resolution can be easily achieved by directly cascading a down-sampling module to the bitstream in the compressed domain.

Furthermore, combining Super-Format with scalable coding can achieve the ultimate media adaptation in which a scalable decoder is embedded in scalable bitstreams to solve the format compatibility issue and the embedded scalable bitstreams is utilized to solve the physical compatibility issue such as bit rate, screen resolution, or the like.

5.2 Complexity balancing between encoder and decoder?

Conventionally, the complexity of the encoder and the corresponding decoder are asymmetric with encoder being multiple folds more complex than the decoder. The design philosophy is correct since in most situations we compress the video only but watch it multiple times or by many other people (i.e., TV broadcast): cheap decoders are highly desirable while expensive encoders are affordable by TV stations. Even though we see the complexity of decoders increasing over generations of video coding standards, the complexity of the corresponding encoders has increased by a far larger margin.

On the other hand, the new distributed video coding (DVC) paradigm has completely shifted the complexity from the encoder to the decoder (i.e., explore the correlation at the decoder instead of the encoder as conventional coding paradigm). The most applicable scenario is that the sources are of restricted computation power, such as a wireless sensor or a mobile phone. Even though theoretical results (i.e., Slepian-Wolf and Wyner-Ziv theorems) advocate that the coding efficiency bound is the same as the conventional coding paradigms, the reported coding results lag far behind while the decoder complexity has already well exceeded that of a conventional encoder (mainly due to the complex channel decoding module). At the time of this writing, a detailed comparison for the complexity of conventional codec and DVC codec is fairly alluring.

The two coding paradigms have exploited the correlation in either the encoder or the decoder, but not both, which has directly led to the extreme asymmetric complexity between encoder and decoder. Therefore, the questions we want to ask are:

- If the coding efficiency bound remains the same, is it possible for us to exploit half of the correlation at the encoder and half at the decoder?
- If we can freely divide the task of correlation exploitation between the encoder and the decoder would there be any optimal point we can achieve so that the coding efficiency is maximized while the overall coding complexity of both the encoder and the decoder are minimized?

VI. CONCLUDING REMARKS

In this article, we use video communication as an example to present an overview of the next generation mobile multimedia communication platform. We have discussed several core enabling technologies that we think would cause a revolution in mobile media space, namely, novel coding technology, efficient and robust transport, and wireless ad hoc networking. New application scenarios and services would occur based on these core technologies, such as, efficient wireless streaming, multiparty video conferencing and live streaming, collaborative streaming, incremental streaming, soft storage capacity, and intelligent synchronization between devices. This article does not attempt to address every aspect of mobile multimedia communications. We are hopeful that this article would trigger some more in-depth thinking and push the state-of-the-art in mobile communication forward.

VI. ACKNOWLEDGEMENT

The authors would like to thank many colleagues at Microsoft Research Asia in providing useful information and insights to complete this article, especially Dr. Jiang Li from the Media Communication group, Dr. Xiaoyan Sun, and Mr. Ji-Zheng Xu from the Internet Media group.

REFERENCES

[1] J. Li, K. Yu, T. He, Y. Lin, S. Li, and Y.-Q. Zhang, Scalable Portrait Video for Mobile Video Communication, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 5, May 2003, pages 376 - 384.

[2] W. Li, "Overview of Fine Granularity Scalability in MPEG-4 video standard", IEEE Trans. Circuits and Systems for Video Technology, vol. 11, no 3, pp. 301-317, 2001.

[3] H. Radha, M. van der Schaar, Y. Chen, "The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming over IP", IEEE Trans. on Multimedia, vol. 3, no 1, pp. 53 - 68, 2001.

[4] F. Wu, S. Li and Y.-Q. Zhang, "DCT-prediction

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1274
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

based progressive fine granularity scalability coding", Proc. IEEE ICIP 2000, vol. 3, pp. 556-559, Canada, 2000.

[5] F. Wu, S. Li and Y.-Q. Zhang, "A framework for efficient progressive fine granularity scalable video coding", IEEE Trans. Circuits and Systems for Video Technology, special issue on streaming video, vol. 11, no 3, pp. 332-344, 2001.

[6] H. Huang, C. Wang, T. Chiang, "A robust fine granularity scalability using trellis-based predictive leak", IEEE Trans. Circuits and Systems for Video Technology, vol. 12, no. 6, pp. 372-385, 2002

[7] A. Fuldseth and T. A. Ramstad, "Robust subband video coding with leaky prediction," in Proc. DSP Workshop, pp. 57-60, Norway, 1996.

[8] X. Sun, F. Wu, S. Li, W. Gao, Y,-Q. Zhang, "Macroblock-based progressive fine granularity scalable video coding", Proc. IEEE ICME, Japan, 2001.

[9] F. Wu, S. Li, X. Sun, B. Zeng and Y.-Q. Zhang, "Macroblock-based progressive fine granularity scalable video coding", International Journal of Imaging Systems and Technology, vol.13, no. 6, pp. 297-307, 2003.

[10] G. Karlsson and M. Vetterli: "Subband coding of video signals for packet switched networks", Proc. Visual Commun. Image Process., SPIE vol. 845, pp. 446-456, 1987.

[11] T. Kronander: "Some aspects of perception based image coding", PhD Thesis, Link?ping University 1989.

[12] J.-R. Ohm: "Three-dimensional subband coding with motion compensation", IEEE Trans. Image Processing, no. 3, pp. 559-571, 1994.

[13] S.-J. Choi and J. W. Woods, "Motion compensated 3D subband coding of video," IEEE Trans. Image Processing, vol. 8, pp. 155-167, 1999.

[14] R. Xiong, F. Wu, J. Xu, S. Li, Y.-Q. Zhang, "Barbell lifting wavelet transform for highly scalable video coding, " Proc. Picture Coding Symposium 2004, San Francisco, CA, USA, Dec 2004.

[15] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," IEEE Trans. on Circ. and Systems for Video Techn., vol. 6, pp. 243-250, June 1996

[16] D. Taubman, "High performance scalable image compression with EBCOT," IEEE Trans. on Image Processing, no. 9, pp. 1158-1170, 2000

[17] S.-T. Hsiang and J. W. Woods, "Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling," MPEG-4 Workshop and Exhibition at ISCAS 2000, Geneva, Switzerland, May 2000

[18] J. Xu, S. Li and Y.-Q. Zhang, "A wavelet video coder using three dimensional embedded subband coding with optimized truncation (3D ESCOT)," IEEE-PCM 2000, Sydney, Dec. 2000

[19] J. Xu, Z. Xiong, S. Li and Y.-Q. Zhang, "3D embedded subband coding with optimal truncation (3D-ESCOT)," Applied and Computational Harmonic Analysis 10, p. 589, May 2001

[20] S.-T. Hsiang and J. W. Woods, "Embedded video coding using invertible motion compensated 3D subband/wavelet filter bank," Signal Processing:, Image Commun.., vol. 16, pp. 705-724, May 2001.

[21] S.-T. Hsiang, J. Woods, and J.-R. Ohm, "Invertible temporal subband/wavelet filter banks with half-pixel accurate motion compensation," IEEE Trans. on Image Processing vol. 13, pp. 1018-1028, August 2004.

[22] C. Boutremans and J.-Y. Le Boudec, "Adaptive joint playout buffer and FEC adjustment for Internet telephony," in Proc. 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '0 3), vol. 1, pp. 652-662, San Francisco, Calif, USA, March-April 2003.

[23] Q. Zhang and S. Kassam, "Hybrid ARQ with selective combining for fading channels," IEEE J. Select. Areas Commun., vol. 17, no. 5, pp. 867-880, 1999.

[24] T. Zhang and Y. Xu, "Unequal packet loss protection for layered video transmission," IEEE Trans. Broadcast., vol. 45, no. 2, pp. 243-252, 1999.

[25] Q. Zhang, G. Wang, W. Zhu, and Y.-Q. Zhang, "Robust scalable video streaming over Internet with network-adaptive congestion control and unequal loss protection," in Proc. Packet Video Workshop, Kyongju, Korea, April 2001.

[26] B. Tao, H. A. Peterson, and B. W. Dickinson, "A ratequantization model for MPEG encoders," in Proc. IEEE International Conference on Image Processing (ICIP 育 7), vol. 1, pp. 338-341, Santa Barbara, Calif, USA, October 1997.

[27] Z. He, J. Cai, and C. Chen, "Joint source channel rate distortion analysis for adaptive mode selection and rate control in wireless video coding," IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 6, pp. 511-523, 2002.

[28] F. Zhai, Y. Eisenberg, T. N. Pappas, R. Berry, and A. K. Katsaggelos, "Rate-distortion optimized hybrid error control for real-time packetized video transmission," in Proc. IEEE International Conference on Communications (ICC '0 4), vol. 3, pp. 1318-1322, Paris, France, June 2004.

[29] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," IEEE/ACM Trans. Networking, vol. 7, no. 4, pp. 458-472, 1999.

[30] M. Allman, V. Paxson, andW. Stevens, "TCP congestion control," IETF RFC 2581, April 1999.

[31] R. Rejaie, M. Handley, and D. Estrin, "Rap: an end-toend rate-based congestion control mechanism for realtime streams in the Internet," in Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99), vol. 3, pp. 1337-1345, New York, NY, USA, March 1999.

[32] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1275
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

"Modeling TCP Reno performance: a simple model and its empirical validation," IEEE/ACMTrans.Networking, vol. 8, no. 2, pp. 133-145, 2000.

[33] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation based congestion control for unicast applications, " in Proc. ACM SIGCOMM 2000, pp. 43-56, Stockholm, Sweden, August 2000, http://www.aciri.org/tfrc.

[34] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in Proc. IEEE 15th International Conference on Distributed Computing Systems (ICDCS '95), pp. 136-143, Vancouver, British Columbia, Canada, May-June 1995.

[35] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," IEEE/ACM Trans. Networking, vol. 5, no. 6, pp. 756-769, 1997.

[36] R. Yavatkar and N. Bhagawat, "Improving end-to-end performance of TCP over mobile internetworks", Proceedings of theWorkshop on Mobile Computing Systems and Applications, Santa Cruz, CA, pp. 146-152.

[37] S. Biaz and N. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver," in Proc. IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET '99), pp. 10-17, Richardson, Tex, USA, March 1999.

[38] S. Cen, P. Cosman, and G. Voelker, "End-to-end differentiation of congestion and wireless losses," in Proc. SPIE Multimedia Computing and Networking (MMCN '02), M. G. Kienzle and P. J. Shenoy, Eds., vol. 4673 of Proceedings of SPIE, pp. 1-15, San Jose, Calif, USA, January 2002.

[39] D. Barman and I. Matta, "Effectiveness of loss labeling in improving TCP performance in wired/wireless networks," in Proc. 10th IEEE International Conference on Network Protocols (ICNP '02), pp. 2-11, Paris, France, November 2002.

[40] K. Tan, Q. Zhang and W. Zhu, "An End-to-end Rate control Protocol for Multimedia Streaming in Wired-cum-wireless Environments", In Proc. ISCAS 2003, vol 2, pp. 836-839.

[41] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," IEEE J. Select. Areas Commun., vol. 21, no. 2, pp. 216-228, 2003.

[42] M. Gerla, B. K. F. Ng, M. Y. Sanadidi, M. Valla, and R. Wang, "TCP westwood with adaptive bandwidth estimation to improve efficiency/friendliness tradeoffs," Computer Communications, vol. 27, no. 1, pp. 41-58, 2004.

[43] Service-Centric Network Setup for Spontaneous Collaboration using Ad hoc Networks, http://research.microsoft.com/wn/ezsetup.aspx.

[44] IEEE Std 802.11-1997, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. June. 1997.

[45] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol. IETF RFC 2165, June 1997.

[46] Universal Plug and Play Device Architecture, http://www.upnp.org/download/UPnPDA10_20000613.htm , version 1.0, June 2000.

[47] F. Wu, H. Sun, G. Shen, S. Li, Y.-Q. Zhang, B. Lin, M.-C. Li, "SMART: an efficient, scalable and robust streaming video system", EURASIP Journal on Applied Signal Processing, Special issue on Multimedia over IP and Wireless Networks, vol. 2, pp 192-206, 2004.

[48] H. Wang, G. Shen, S. Li and Y. Zhong, "Enhancing Multimedia Streaming Performance Through Peer-Paired Collaboration," Proc. of 2002 IEEE International Conference on Image Processing (ICIP-2002), Rochester, New York, vol. 3, Sep. 22-25, 2002.

[49] H. Wang, G. Shen, S. Li and Y. Zhong, "Efficient Error Recovery Techniques in a Novel Multimedia Streaming Framework with Peer-paired Collaboration," Proc. of 2002 IEEE International Symposium on Circuits and Systems (ISCAS-2002) , pp. 393-396.

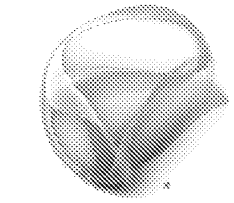
[50] J. Xin, C.-W. Lin, and M.-T. Sun, "Digital video Transcoding," Proceedings of the IEEE, vol. 53, no. 1, pp. 84-97, Jan. 2005.

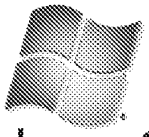
[51] A. Vetro, C. Christopoulos, and H. Sun, "Video Transcoding Architectures and Techniques: An Overview, " IEEE Signal Processing Magazine, vol. 20, no. 2, pp. 18-29, March. 2003.

[52] J. Xu, F. wu, and S. Li, "Transcoding for Progressive Fine Granularity Scalable Video Coding," in Proceedings of IEEE International Symposium on Cirsuits and Systems, 2004, pp. 765-768.

[53] G. Shen, Y. He, W. Cao and S. Li, "MPEG-2 to WMV Transcoder with Adaptive Error Compensation and Dynamic Switches", accepted by IEEE Trans. On Circuits Systems for Video Technology.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1276
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)





Windows Server
Internet Information Services 7.0

IIS Smooth Streaming Technical Overview

Author:

Alex Zambelli, Media Technology Evangelist

Microsoft Corporation – March 2009

Contents

Introduction 3

A Brief History of Multiple Bit Rate Streaming 3

Windows Media Stream Thinning, MBR and Intelligent Streaming 3

The Shift to HTTP-Based Delivery 4

Content Delivery Techniques 5

Traditional Streaming 5

Progressive Download 6

HTTP-Based Adaptive Streaming 7

Introducing IIS Smooth Streaming 9

Smooth Streaming Playback with Silverlight 10

Smooth Streaming Architecture 11

Introducing the Smooth Streaming Format 11

Smooth Streaming Disk File Format 11

Smooth Streaming Wire File Format 12

Smooth Streaming Media Assets 13

Smooth Streaming Manifest Files 14

Smooth Streaming Playback 15

Summary 16

For More Information 16

Legal Notice 17

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1278
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Introduction

In October 2008, Microsoft announced that Internet Information Services (IIS) 7.0 would feature a new HTTP-based adaptive streaming extension: **Smooth Streaming**. To promote the new technology, Microsoft also announced an initiative with Akamai and launched a showcase Web site—SmoothHD.com.

Other content delivery networks (CDNs) are expected to announce support for Smooth Streaming in the future.

Smooth Streaming dynamically detects local bandwidth and CPU conditions and seamlessly switches, in near real time, the video quality of a media file that a player receives. Consumers with high-bandwidth connections can experience high definition (HD) quality streaming while others with lower bandwidth speeds receive the appropriate stream for their connectivity, allowing consumers across the board to enjoy a compelling, uninterrupted streaming experience and alleviating the need for media companies to cater to the lowest common denominator quality level within their audience base.

This enables companies to boost brand awareness and advertising revenues by extending average viewing times through higher quality true HD (resolution greater than 720p) experiences. They can also benefit from unprecedented network scalability using distributed HTTP-based Web servers and offer better quality to more customers.

To understand how Smooth Streaming works and what its benefits are, we must first take a look at the history of multiple bit rate media streaming on the Web.

A Brief History of Multiple Bit Rate Streaming

Windows Media Stream Thinning, MBR and Intelligent Streaming

The first effort to adapt streams to client conditions was called *stream thinning*, which Microsoft introduced in 1998 as part of NetShow Services 3.0 and Windows Media® Player 6.1. Stream thinning automatically detected deteriorating network conditions and decreased the video frame rate in response. In dire network conditions, the client could even suspend video playback entirely and stream only audio.

The earliest form of multiple bit rate streaming was introduced by Microsoft in 2000, as part of Windows Media Technologies 4.0 and Windows Media Services 4.1 (in Windows® 2000 Server/Windows NT® Server 4.0) together with Windows Media Player 6.4. The ASF file format allows storage of multiple video and audio tracks inside a single file, and the Windows Media streaming protocols support switching streams during a broadcast. This technology is most commonly referred to as *Multiple Bit Rate ASF*, or simply *MBR*.

In 2002, Microsoft released the Windows Media 9 Series products. Windows Media Services 9 Series (in Windows Server® 2003) and Windows Media Player 9 Series introduced an improved MBR technology dubbed *Intelligent Streaming*. Intelligent Streaming combined bandwidth detection, stream thinning,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1279
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

MBR ASF, and better image handling in Windows Media Player. Intelligent Streaming, of course, still required the media to be encoded as MBR ASF files with a tool such as Windows Media Encoder 9 Series.

While the technology itself was well designed, its implementations suffered from shortcomings. It was limited to streaming only (no progressive download), and only from Windows Media servers. The encoders did not require that the multiple video streams be temporally aligned, let alone key-frame aligned, which made switching between streams difficult to do in a seamless fashion. Because the media was streamed, and streaming protocols function at constant rates, it was almost impossible to accurately predict overall client bandwidth—particularly in a timely fashion. By the time poor network conditions were detected, it was usually already too late—the Player often went through several iterations of re-buffering before finally downgrading the bit rate.

The Shift to HTTP-Based Delivery

One of the trends that have emerged in the streaming media industry over the past several years has been a steady shift away from classic streaming protocols (RTSP, MMS, RTMP, etc.) back to plain HTTP download. There are many examples of this trend today with community video sharing sites that employ only HTTP progressive download for media delivery.

There are several strong reasons for this industry trend:

- Web download services have traditionally been less expensive than media streaming services offered by CDNs and hosting providers.
- Media protocols often have difficulty getting around firewalls and routers because they are commonly based on UDP sockets over unusual port numbers. HTTP-based media delivery has no such problems because firewalls and routers know to pass HTTP downloads through port 80.
- HTTP media delivery doesn't require special proxies or caches. A media file is just like any other file to a Web cache.
- It's much easier and cheaper to move HTTP data to the edge of the network, closer to users.

Even though streaming protocols are designed with media delivery in mind, the fact of the matter is that the Internet was built on HTTP and optimized for HTTP delivery. Consequently this begged the question, "Why not adapt media delivery to the Internet instead of trying to adapt the entire Internet to streaming protocols?"

Move Networks proved on several occasions in 2008 that HTTP-based media delivery could be done successfully on a large scale—both on-demand and live, such as with the broadcast of the Democratic National Convention , using Microsoft® Silverlight™ as the client framework. Microsoft demonstrated this again during coverage of the 2008 Beijing Summer Olympic Games, when it prototyped its own HTTP-based adaptive streaming platform.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1280
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Content Delivery Techniques

Media delivery on the Web today uses three general delivery methods: traditional streaming, progressive download, and adaptive streaming.

Traditional Streaming

RTSP (Real-Time Streaming Protocol) is a good example of a traditional streaming protocol. RTSP is defined as a *stateful* protocol, which means that from the first time a client connects to the streaming server until the time it disconnects from the streaming server, the server keeps track of the client's state. The client communicates its state to the server by issuing it commands such as PLAY, PAUSE or TEARDOWN (the first two are obvious; the last one is used to disconnect from the server and close the streaming session).

After a session between the client and the server has been established, the server begins sending the media as a steady stream of small packets (the format of these packets is known as RTP). The size of a typical RTP packet is 1452 bytes, which means that in a video stream encoded at 1 megabits per second (Mbps), each packet carries approximately 11 milliseconds of video. In RTSP the packets can be transmitted over either UDP or TCP transports—the latter is preferred when firewalls or proxies block UDP packets, but can also lead to increased latency (TCP packets are re-sent until received).

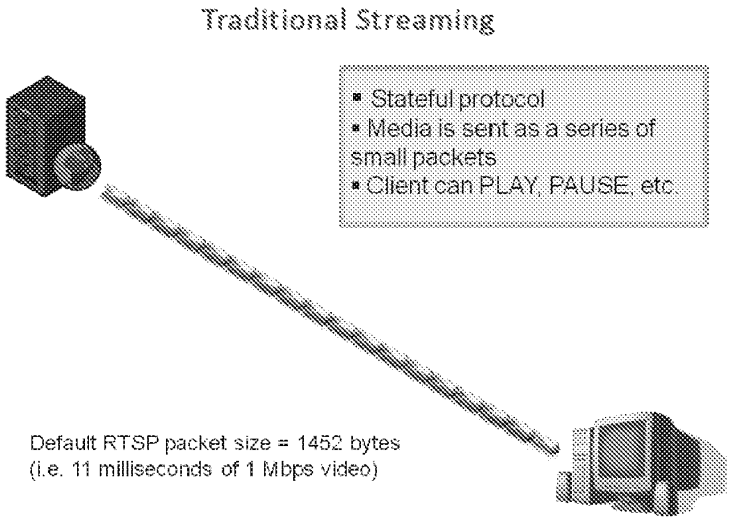


Figure 1. RTSP is an example of a traditional streaming protocol.

HTTP, on the other hand, is known as a *stateless* protocol. If an HTTP client requests some data, the server responds by sending the data, but it won't remember the client or its state. Each HTTP request is handled as a completely standalone one-time session.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1281
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Windows Media Services supports streaming over both RTSP and HTTP. But if HTTP is a stateless protocol, how can it be used for streaming? Windows Media Services uses a modified version of HTTP officially known as MS-WMSP (known in Windows Media Services as the Windows Media HTTP Streaming Protocol, or more commonly just as Windows Media HTTP). MS-WMSP uses standard HTTP for transfer of data and messages but also maintains session states, effectively turning it into a streaming protocol like RTSP. Windows Media Services has also supported RTSP streaming since 2003 (in Windows Media Services 9 Series) over both UDP and TCP. Its implementation of the protocol is publicly documented as MS-RTSP.

Silverlight only supports HTTP-based delivery from Windows Media Services.

The most important things to remember about traditional streaming protocols such as RTSP and Windows Media HTTP (MS-WMSP) are:

- The server sends the data packets to the client at a real-time rate only—that is, the bit rate at which the media is encoded. For example, a video encoded at 500 kilobits per second (kbps) is streamed to clients at approximately 500 kbps.
- The server only sends ahead enough data packets to fill the client buffer. The client buffer is typically between 1 and 10 seconds (Windows Media Player and Silverlight default buffer length is 5 seconds). This means that if you pause a streamed video and wait 10 minutes, still only approximately 5 seconds of video will have downloaded to the client in that time.

Other examples of traditional streaming protocols include Adobe Systems' proprietary Real Time Messaging Protocol (RTMP) and RealNetworks' RTSP over Real Data Transport (RDT) protocol. The Dynamic Streaming stream-switching feature in the Adobe® Flash® Platform is based on the RTMP protocol and is, therefore, considered a traditional streaming method—not adaptive streaming.

Progressive Download

Another common form of media delivery on the Web today is progressive download, which is nothing more than a simple file download from an HTTP Web server. Progressive download is supported by most media players and platforms, including Adobe Flash, Silverlight, and Windows Media Player. The term "progressive" stems from the fact that most player clients allow the media file to be played back while the download is still in progress—before the entire file has been fully written to disk (typically to the Web browser cache). Clients that support the HTTP 1.1 specification can also seek to positions in the media file that haven't been downloaded yet by performing byte range requests to the Web server (assuming that it also supports HTTP 1.1).

Popular video sharing Web sites on the Web today, including YouTube, Vimeo, MySpace, and MSN Soapbox, almost exclusively use progressive download.

Unlike streaming servers that rarely send more than 10 seconds of media data to the client at a time, HTTP Web servers keep the data flowing until the download is complete. If you pause a progressively downloaded video at the beginning of playback and then wait, the entire video will eventually have downloaded to your browser cache, allowing you to smoothly play the whole video without any hiccups.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1282
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

There is a downside to this behavior as well—if 30 seconds into a fully downloaded 10 minute video, you decide that you don't like it and quit the video, both you and your content provider have just wasted 9 minutes and 30 seconds worth of bandwidth. To try to mitigate this problem, IIS 7.0 provides a cool extension called Bit Rate Throttling, which allows content providers to throttle the download bit rate in exactly the same way that a streaming server would to reduce costs.

HTTP-Based Adaptive Streaming

Adaptive streaming is a hybrid delivery method that acts like streaming but is based on HTTP progressive download. It's an advanced concept that uses HTTP rather than a new protocol. Both IIS Smooth Streaming and Move Networks Adaptive Stream are examples of adaptive streaming. Even though the two technologies use different codecs, formats, and encryption schemes, they both rely on HTTP as the transport protocol and perform the media download as a long series of very small progressive downloads, rather than one big progressive download.

In a typical adaptive streaming implementation, the video/audio source is cut into many short segments ("chunks") and encoded to the desired delivery format. Chunks are typically 2-to-4-seconds long. At the video codec level, this typically means that each chunk is cut along video GOP (Group of Pictures) boundaries (each chunk starts with a key frame) and has no dependencies on past or future chunks/GOPs. This allows each chunk to later be decoded independently of other chunks.

The encoded chunks are hosted on a HTTP Web server. A client requests the chunks from the Web server in a linear fashion and downloads them using plain HTTP progressive download. As the chunks are downloaded to the client, the client plays back the sequence of chunks in linear order. Because the chunks are carefully encoded without any gaps or overlaps between them, the chunks play back as a seamless video.

The "adaptive" part of the solution comes into play when the video/audio source is encoded at multiple bit rates, generating multiple chunks of various sizes for each 2-to-4-seconds of video. The client can now choose between chunks of different sizes. Because Web servers usually deliver data as fast as network bandwidth allows them to, the client can easily estimate user bandwidth and decide to download larger or smaller chunks ahead of time. The size of the playback/download buffer is fully customizable.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1283
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

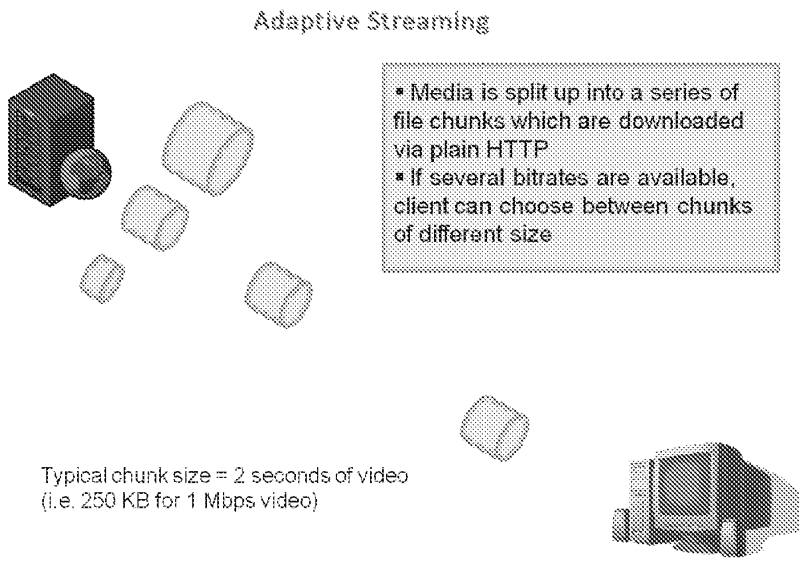


Figure 2. Adaptive streaming is a hybrid media delivery method.

Adaptive streaming, like other forms of HTTP delivery, offers the following advantages over traditional streaming to the content distributor:

- It's cheaper to deploy because adaptive streaming can use generic HTTP caches/proxies and doesn't require specialized servers at each node.
- It offers better scalability and reach, reducing "last mile" issues because it can dynamically adapt to inferior network conditions as it gets closer to the user's home.
- It lets the audience adapt to the content, rather than requiring content providers to guess which bit rates are most likely to be accessible to their audience.

It also offers the following benefits for the user:

- Fast start-up and seek times because start-up/seeking can be initiated on the lowest bit rate before moving to a higher bit rate.
- No buffering, no disconnects, no playback stutter (as long as the user meets the minimum bit rate requirement).
- Seamless bit rate switching based on network conditions and CPU capabilities.
- A generally consistent, **smooth** playback experience.

Microsoft created a prototype implementation of HTTP-based adaptive streaming for the NBC 2008 Beijing Summer Olympic Games Web site. To meet the project's rapid development schedule, this implementation was very straightforward. NBC used Digital Rapids and Anystream encoders to produce multiple Windows Media Video (WMV) files of different bit rates/resolutions for each source. The

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1284
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

encoders didn't employ any new encoding tricks but merely followed strict encoding guidelines (closed GOP, fixed-length GOP, VC-1 entry point headers, and so on.) which ensured exact frame alignment across the various bit rates of the same video. These WMV files were run through a post-processing tool that physically split each WMV file into thousands of 2-second chunks (files). The rest of the solution consisted of uploading the chunks to the CDN's Web servers and then building a Silverlight player that would download the chunks and play them in sequence.

With this implementation, NBC and Microsoft were able to offer a better-than-WMS streaming experience while using just simple HTTP download, with increased average content viewing times that directly translated to better advertising and monetization opportunities.

However, CDN operators lost many hours managing the millions of tiny files in their systems. Imagine: if each 2-seconds of video is split into a separate file and this is repeated for 5 available bit rates, you end up with 150 files for each minute of video. That's 13,500 files for a 90-minute soccer game!

So despite the NBC Olympics site being a huge success for Silverlight and HTTP-based adaptive streaming, it quickly became apparent that to productize this solution and offer improved file-management benefits, elementary design changes were required.

Introducing IIS Smooth Streaming

The IIS Media team productized the adaptive streaming solution used for the NBC Olympics site into a real server product. Its official name is **IIS Smooth Streaming**, an extension for Internet Information Services 7.0.

The IIS Media team redesigned the content creation and delivery aspects of the Olympic Games prototype to improve file management while retaining advantages from the original solution. The new design eschewed the one-file-per-chunk approach in favor of a single contiguous file for each encoded bit rate. The file format of choice would be MPEG-4.

IIS Smooth Streaming uses the MPEG-4 Part 14 (ISO/IEC 14496-12) file format as its disk (storage) and wire (transport) format. Specifically, the Smooth Streaming specification defines each chunk/GOP as an MPEG-4 Movie Fragment and stores it within a contiguous MP4 file for easy random access. One MP4 file is expected for each bit rate. When a client requests a specific source time segment from the IIS Web server, the server dynamically finds the appropriate Movie Fragment box within the contiguous MP4 file and sends it over the wire as a standalone file, thus ensuring full cacheability downstream.

In other words, with Smooth Streaming, file chunks are created virtually upon client request, but the actual video is stored on disk as a single full-length file per encoded bit rate. This offers tremendous file-management benefits.

You can see Smooth Streaming in action at http://www.smoothhd.com. The Smooth Streaming extension for IIS 7.0 is available from The Official Microsoft IIS Web site.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1285
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

On the content-creation end, encoding on-demand Smooth Streaming-compatible video is already possible with Expression Encoder 2 Service Pack 1 (SP1). Note that you'll need to purchase the full version of Expression Encoder 2 to get Smooth Streaming encoding support—it's not included in the "Express" trial version. We recommend the Smooth Streaming Multi Bit Rate Calculator as a helper tool for encoding to multiple bit rate formats such as Smooth Streaming.

In addition, Microsoft is working with a number of encoding independent software vendors (ISVs) to enable support for the Smooth Streaming format in their professional encoding products.

Smooth Streaming Playback with Silverlight

Silverlight 2 already supports playback of Smooth Streaming sources. But how does it do it? Silverlight support for Smooth Streaming, including the parsing of the MPEG-4 file format, the HTTP download, the bit rate switching heuristics, etc., is provided in .NET code. This allows developers to modify and fine-tune the client adaptive streaming code as needed, instead of waiting for the next Silverlight release.

The most challenging part of Smooth Streaming Silverlight client development is the heuristics module that determines when and how to switch bit rates. Elementary stream switching functionality requires the ability to swiftly adapt to changing network conditions while not falling too far behind, but that's often not enough to deliver a great experience. One must also consider:

- What if the user has enough bandwidth but doesn't have enough CPU power to consume the high bit rates/resolutions?
- What happens when the video is paused or hidden in the background (minimized browser window)?
- What if the resolution of the best available video stream is actually larger than the screen resolution, thus wasting bandwidth?
- How large should the download buffer window be?
- How does one ensure seamless rollover to new media assets such as advertisements?

As any Web application developer will tell you, there's much more to building a good player than just setting a source URL for the media element.

Fortunately for those who prefer not to write such code from scratch, there are two options already available for adding Smooth Streaming support to your Silverlight application:

- **Expression Encoder 2 SP1 templates**. The Silverlight 2 player templates included with Expression® Encoder 2 SP1 include a ready-to-use Smooth Streaming module and complete source code (which, if needed, allows fine-tuning of the switching heuristics to adjust to particular needs of a given network or device). The Smooth Streaming object (AdaptiveStreaming.dll) can be easily integrated into any Silverlight project. See James Clarke's Weblog for additional Expression Encoder tips & tricks.
- **Open Video Player (OVP)**. The Akamai-led Open Video Player Initiative is an open-source community project that strives to provide a best-of-breed video player platform for Silverlight and Flash. The Silverlight version of the Open Video Player provides integrated support for

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1286
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Smooth Streaming playback, and is the video player used by Akamai on SmoothHD.com and many of their customer sites.

Smooth Streaming Architecture

Introducing the Smooth Streaming Format

Smooth Streaming is the first Microsoft media format in over a decade to use a file format other than ASF. It's based on the ISO/IEC 14496-12 ISO Base Media File Format specification, better known as the MP4 file specification. Why MP4 and not ASF? Well, there are several reasons:

- MP4 is a lightweight container format with less overhead than ASF.
- MP4 is easier to parse in managed (.NET) code than ASF.
- MP4 is based on a widely used standard, making 3rd party adoption and support more straightforward.
- MP4 is architected with H.264 video codec support in mind. H.264 is an industry leading video compression standard that has been adopted across a broad range of Microsoft products, including Silverlight 3, Windows 7, Xbox 360, Zune and MediaRoom.
- MP4 is designed to natively support payload fragmentation within the file.

There are actually two parts to the Smooth Streaming format: the wire format, and the disk file format. In Smooth Streaming, a video is recorded in full length to the disk as a single file (one file per encoded bit rate), but it's transferred to the client as a series of small file chunks. The wire format defines the structure of the chunks that are sent by IIS to the client, whereas the file format defines the structure of the contiguous file on disk, enabling better file management. Fortunately, the MP4 specification allows MP4 to be internally organized as a series of fragments, which means that in Smooth Streaming the wire format is a direct subset of the file format.

Smooth Streaming Disk File Format

The basic unit of an MP4 file is called a "box." These boxes can contain both data and metadata. The MP4 specification allows for various ways to organize data and metadata boxes within a file. In most media scenarios, it's considered useful to have the metadata written before the data so that a player client application can have more information about the video/audio that it's about to play before it plays it. However, in live streaming scenarios it's often not possible to write the metadata up-front about the whole data stream because it's simply not yet fully known. Furthermore, less up-front metadata means less overhead, which can lead to shorter startup times. For these reasons the MP4 ISO Base Media File Format specification is designed to allow MP4 boxes to be organized in a fragmented manner, where the file can be written "as you go" as a series of short metadata/data box pairs, rather than one long metadata/data pair. The Smooth Streaming file format heavily leverages this aspect of the MP4 file specification, to the point where at Microsoft we often interchangeably refer to Smooth Streaming files as "Fragmented MP4 files" or "(f)MP4."

The following figure is a high-level overview of what a Smooth Streaming file looks like on the inside:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1287
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

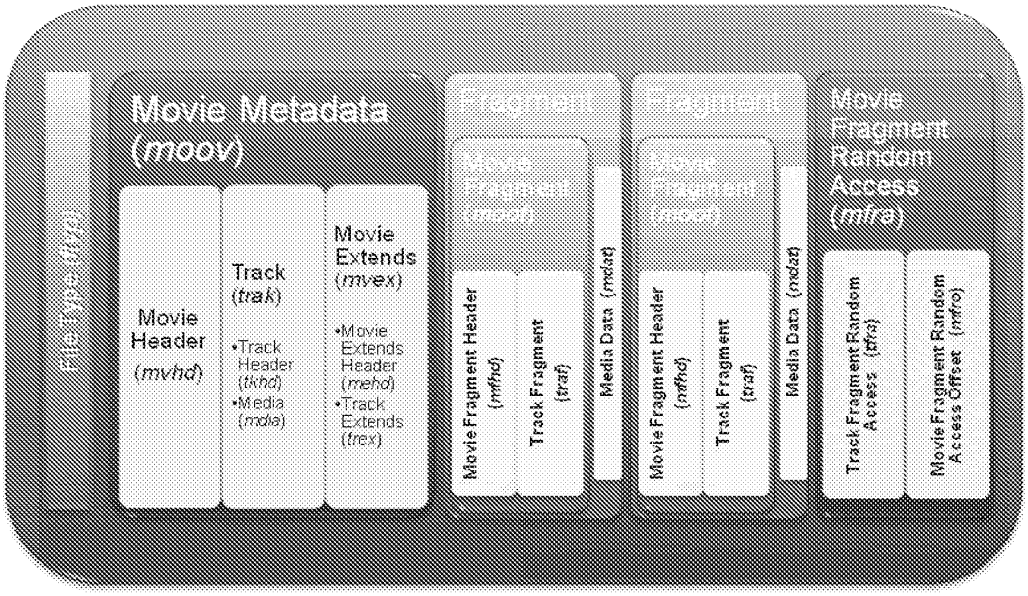


Figure 3. Smooth Streaming File Format

In a nutshell, the file starts with file-level metadata ('moov') that generically describes the file, but the bulk of the payload is actually contained in the fragment boxes that also carry more accurate fragment-level metadata ('moof') and media data ('mdat'). (The diagram in Figure 3 only shows 2 fragments, but a typical Smooth Streaming file has a fragment for each 2 seconds of video/audio.) Closing the file is an 'mfra' index box that allows easy and accurate seeking within the file.

Smooth Streaming Wire File Format

When a player client requests a video time slice from the IIS Web server, the server seeks to the appropriate starting fragment in the MP4 file and then lifts the fragment out of the file and sends it over the wire to the client. This is why we refer to the fragments as the "wire format." This technique greatly enhances the efficiency of the IIS Web server because it doesn't induce any re-muxing or rewriting overhead.

The following figure shows what an MP4 fragment looks like in more detail:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1288
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

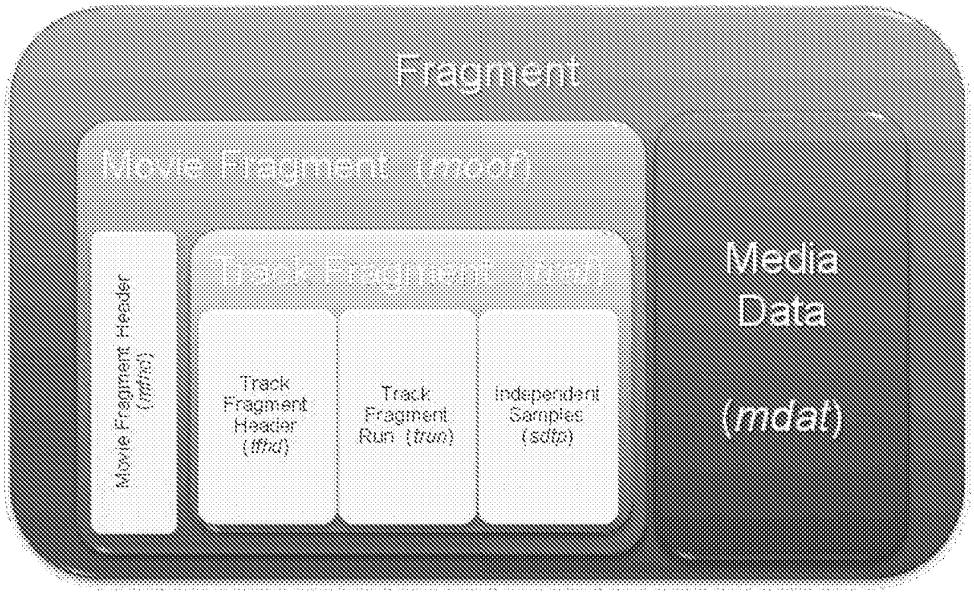


Figure 4. Smooth Streaming Wire Format.

Within the guidelines of the MP4 ISO Base Media File Format specification, the Smooth Streaming format uses a custom box organization schema and some custom boxes. To differentiate Smooth Streaming files from "vanilla" MP4 files, we use new file extensions: ***.ismv** (video+audio) and ***.isma** (audio only).

Smooth Streaming Media Assets

A typical Smooth Streaming media asset (presentation) consists of the following files:

- MP4 files containing video/audio
  - ***.ismv**
    - Contains video and audio, or only video
    - 1 ISMV file per encoded video bit rate
  - ***.isma**
    - Contains only audio
    - In videos with audio, the audio track can be muxed into an ISMV file instead of a separate ISMA file
- Server manifest file
  - ***.ism**
    - Describes the relationships between the media tracks, bit rates and files on disk
    - Only used by the IIS Smooth Streaming server—not by clients
- Client manifest file
  - ***.ismc**

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1289
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Describes the available streams to the client: the codecs used, bit rates encoded, video resolutions, markers, captions, etc.
- It's the first file delivered to the client

The manifest files are XML-formatted files. The server manifest file format is based specifically on the SMIL 2.0 XML format specification.

A folder containing a single Smooth Streaming presentation might look something like the following:

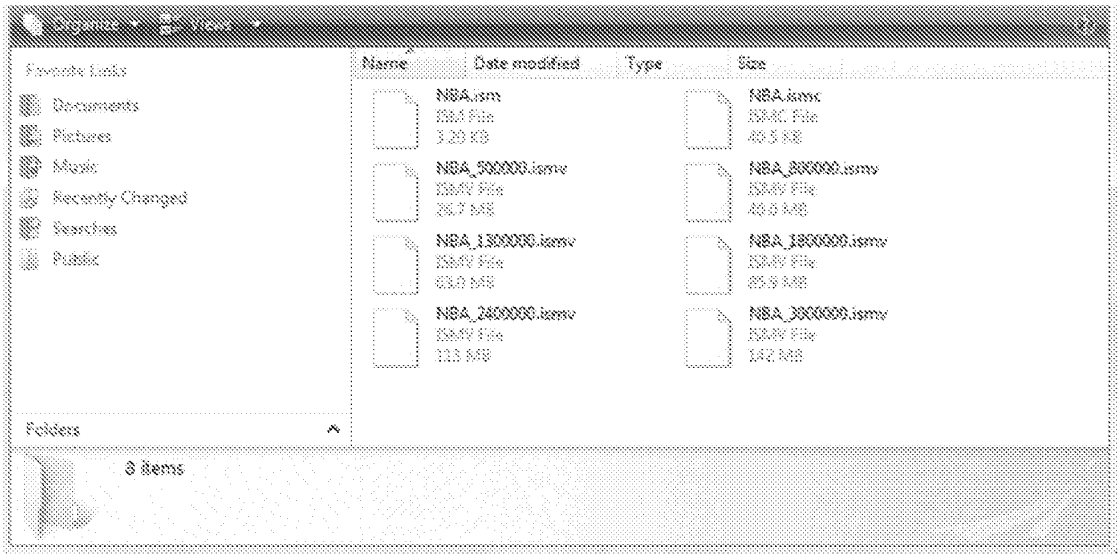


Figure 5. A folder containing a Smooth Streaming encoded presentation. In this particular case, the audio track is contained in the NBA_3000000.ismv file.

Smooth Streaming Manifest Files

The Smooth Streaming Wire/File Format specification defines the manifest XML language as well as the MP4 box structure. Because the manifests are based on XML, they are highly extensible. Among the features already included in the current Smooth Streaming format specification is support for the following:

- VC-1, WMA, H.264 and AAC codecs
- Text streams
- Multi-language audio tracks
- Alternate video and audio tracks (for example, multiple camera angles, director's commentary, etc.)
- Multiple hardware profiles (for example, a bit rate targeted at different playback devices)
- Script commands, markers/chapters, captions
- Client manifest Gzip compression

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1290
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- URL obfuscation
- Live encoding and streaming

Sample Smooth Streaming on-demand server manifest files (.ism) and Smooth Streaming client manifest files (.ismc) are included in the IIS Smooth Streaming Beta Sample Content for IIS Smooth Streaming Beta.

Smooth Streaming Playback

Microsoft's adaptive streaming prototype (used for NBC Olympics 2008) relied on physically chopping up long video files into small file chunks. To retrieve the chunks for the Web server, the player client simply needed to download the files in a logical sequence: 00001.vid, 00002.vid, 00003.vid, and so on.

Smooth Streaming uses a more sophisticated file format and server design. The videos are no longer split up into thousands of file chunks, but are instead "virtually" split into fragments (typically one fragment per video GOP) and stored within a single contiguous MP4 file. This implies two significant changes in server and client design:

- The server must translate URL requests into exact byte range offsets within the MP4 file
- The client can request chunks in a more developer-friendly manner (for example, by timecode instead of by index number)

The first thing a player client requests from the Smooth Streaming server is the ***.ismc** client manifest. The manifest tells it which codecs were used to compress the content (so that the client runtime can initialize the correct decoder and build the playback pipeline), which bit rates and resolutions are available, and a list of the available chunks (with either their start times or durations).

With IIS Smooth Streaming, clients request fragments in the form of a RESTful URL:

- http://video.foo.com/NBA.ism/QualityLevels(400000)/Fragments(video=610275114)

The values passed in the URL represent encoded bit rate (400000) and the fragment start offset (610275114) expressed in an agreed-upon time unit (usually 100 nanoseconds (ns)). These values are known from the client manifest.

After receiving the client request, IIS Smooth Streaming looks up the quality level (bit rate) in the corresponding *.ism server manifest and maps it to a physical *.ismv or *.isma file on disk. It then reads the appropriate MP4 file, and based on its 'tfra' index box, figures out which fragment box ('moof' + 'mdat') corresponds to the requested start time offset. It then extracts the fragment box and sends it over the wire to the client as a standalone file. This is a particularly important part of the overall design because the sent fragment/file can now be automatically cached further down the network, potentially saving the origin server from sending the same fragment/file again to another client that requests the same RESTful URL.

Requesting chunks of video/audio from the server is easy. But what about the bit-rate switching that makes adaptive streaming so effective? This part of the Smooth Streaming experience is implemented

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1291
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

entirely in client-side Silverlight application code—the server plays no part in the bit-rate switching process. The client-side code looks at chunk download times, buffer fullness, rendered frame rates, and other factors, and decides when to request higher or lower bit rates from the server. Remember, if during the encoding process we ensure that all bit rates of the same source are perfectly frame-aligned (same length GOPs, no dropped frames, etc.), then switching between bit rates is completely seamless.

Summary

Smooth Streaming is Microsoft's implementation of HTTP-based adaptive streaming, which is a hybrid media delivery method. It acts like streaming, but is based on HTTP progressive download. The HTTP downloads are performed in a series of small chunks, allowing the media to be easily and cheaply cached along the edge of the network, closer to clients. Providing multiple encoded bit rates of the same media source also allows clients to seamlessly and dynamically switch between bit rates depending on network conditions and CPU power. The resulting user experience is one of reliable, consistent playback without stutter, buffering or "last mile" congestion. In one word: **Smooth**.

For More Information

Media on IIS 7.0	http://iis.net/media
Microsoft Silverlight—Media	http://www.microsoft.com/silverlight/overview/media.aspx
Microsoft Expression	http://www.microsoft.com/expression
Silverlight Developer Center	http://msdn.microsoft.com/silverlight
Microsoft Silverlight Community	http://silverlight.net/community/
Windows Media Services 2008	http://www.microsoft.com/windows/windowsmedia/forpros/serve/prodinfo2008.aspx

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1292
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Legal Notice

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Expression, Silverlight, the Silverlight logo, Windows, the Windows logo, Windows Media, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1293
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

MMUSIC Working Group
Internet-Draft
Obsoletes: 2326 (if approved)
Intended status: Standards Track
Expires: September 10, 2011

H. Schulzrinne
Columbia University
A. Rao
Cisco
R. Lanphier

M. Westerlund
Ericsson AB
M. Stiemerling (Ed.)
NEC

March 9, 2011

Real Time Streaming Protocol 2.0 (RTSP)
draft-ietf-mmusic-rfc2326bis-27

Abstract

This memorandum defines RTSP version 2.0 which obsoletes RTSP version 1.0 which is defined in RFC 2326.

The Real Time Streaming Protocol, or RTSP, is an application-level protocol for setup and control of the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions, provide a means for choosing delivery channels such as UDP, multicast UDP and TCP, and provide a means for choosing delivery mechanisms based upon RTP (RFC 3550).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2011.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1294
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1295
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Table of Contents

1. Introduction 11
2. Protocol Overview 13
2.1. Presentation Description 13
2.2. Session Establishment 14
2.3. Media Delivery Control 15
2.4. Session Parameter Manipulations 17
2.5. Media Delivery 17
2.5.1. Media Delivery Manipulations 18
2.6. Session Maintenance and Termination 20
2.7. Extending RTSP 21
3. Document Conventions 23
3.1. Notational Conventions 23
3.2. Terminology 23
4. Protocol Parameters 27
4.1. RTSP Version 27
4.2. RTSP IRI and URI 27
4.3. Session Identifiers 29
4.4. SMPTE Relative Timestamps 29
4.5. Normal Play Time 30
4.6. Absolute Time 31
4.7. Feature-Tags 31
4.8. Message Body Tags 31
4.9. Media Properties 32
4.9.1. Random Access and Seeking 33
4.9.2. Retention 33
4.9.3. Content Modifications 34
4.9.4. Supported Scale Factors 34
4.9.5. Mapping to the Attributes 34
5. RTSP Message 35
5.1. Message Types 35
5.2. Message Headers 35
5.3. Message Body 36
5.4. Message Length 36
6. General Header Fields 38
7. Request 39
7.1. Request Line 39
7.2. Request Header Fields 41
8. Response 43
8.1. Status-Line 43
8.1.1. Status Code and Reason Phrase 43
8.2. Response Headers 46
9. Message Body 48
9.1. Message-Body Header Fields 48
9.2. Message Body 49
10. Connections 50
10.1. Reliability and Acknowledgements 50

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1296
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

10.2. Using Connections 51
10.3. Closing Connections 53
10.4. Timing Out Connections and RTSP Messages 54
10.5. Showing Liveness 55
10.6. Use of IPv6 56
10.7. Overload Control 56
11. Capability Handling 58
12. Pipelining Support 60
13. Method Definitions 61
13.1. OPTIONS 62
13.2. DESCRIBE 63
13.3. SETUP 65
13.3.1. Changing Transport Parameters 68
13.4. PLAY 69
13.4.1. General Usage 69
13.4.2. Aggregated Sessions 73
13.4.3. Updating current PLAY Requests 74
13.4.4. Playing On-Demand Media 77
13.4.5. Playing Dynamic On-Demand Media 77
13.4.6. Playing Live Media 77
13.4.7. Playing Live with Recording 78
13.4.8. Playing Live with Time-Shift 79
13.5. PLAY_NOTIFY 79
13.5.1. End-of-Stream 80
13.5.2. Media-Properties-Update 81
13.5.3. Scale-Change 82
13.6. PAUSE 84
13.7. TEARDOWN 86
13.7.1. Client to Server 86
13.7.2. Server to Client 87
13.8. GET_PARAMETER 88
13.9. SET_PARAMETER 89
13.10. REDIRECT 91
14. Embedded (Interleaved) Binary Data 94
15. Status Code Definitions 96
15.1. Success 1xx 96
15.1.1. 100 Continue 96
15.2. Success 2xx 96
15.2.1. 200 OK 96
15.3. Redirection 3xx 96
15.3.1. 301 Moved Permanently 97
15.3.2. 302 Found 97
15.3.3. 303 See Other 97
15.3.4. 304 Not Modified 97
15.3.5. 305 Use Proxy 98
15.4. Client Error 4xx 98
15.4.1. 400 Bad Request 98
15.4.2. 401 Unauthorized 98

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1297
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15.4.3. 402 Payment Required 99
15.4.4. 403 Forbidden 99
15.4.5. 404 Not Found 99
15.4.6. 405 Method Not Allowed 99
15.4.7. 406 Not Acceptable 99
15.4.8. 407 Proxy Authentication Required 100
15.4.9. 408 Request Timeout 100
15.4.10. 410 Gone 100
15.4.11. 411 Length Required 100
15.4.12. 412 Precondition Failed 101
15.4.13. 413 Request Message Body Too Large 101
15.4.14. 414 Request-URI Too Long 101
15.4.15. 415 Unsupported Media Type 101
15.4.16. 451 Parameter Not Understood 101
15.4.17. 452 reserved 101
15.4.18. 453 Not Enough Bandwidth 102
15.4.19. 454 Session Not Found 102
15.4.20. 455 Method Not Valid in This State 102
15.4.21. 456 Header Field Not Valid for Resource 102
15.4.22. 457 Invalid Range 102
15.4.23. 458 Parameter Is Read-Only 102
15.4.24. 459 Aggregate Operation Not Allowed 102
15.4.25. 460 Only Aggregate Operation Allowed 102
15.4.26. 461 Unsupported Transport 103
15.4.27. 462 Destination Unreachable 103
15.4.28. 463 Destination Prohibited 103
15.4.29. 464 Data Transport Not Ready Yet 103
15.4.30. 465 Notification Reason Unknown 103
15.4.31. 466 Key Management Error 103
15.4.32. 470 Connection Authorization Required 104
15.4.33. 471 Connection Credentials not accepted 104
15.4.34. 472 Failure to establish secure connection 104
15.5. Server Error 5xx 104
15.5.1. 500 Internal Server Error 104
15.5.2. 501 Not Implemented 104
15.5.3. 502 Bad Gateway 104
15.5.4. 503 Service Unavailable 105
15.5.5. 504 Gateway Timeout 105
15.5.6. 505 RTSP Version Not Supported 105
15.5.7. 551 Option not supported 105
16. Header Field Definitions 106
16.1. Accept 116
16.2. Accept-Credentials 116
16.3. Accept-Encoding 117
16.4. Accept-Language 118
16.5. Accept-Ranges 119
16.6. Allow 119
16.7. Authorization 119

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1298
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.8. Bandwidth 120
16.9. Blocksize 121
16.10. Cache-Control 121
16.11. Connection 123
16.12. Connection-Credentials 124
16.13. Content-Base 125
16.14. Content-Encoding 125
16.15. Content-Language 126
16.16. Content-Length 127
16.17. Content-Location 127
16.18. Content-Type 128
16.19. CSeq 128
16.20. Date 129
16.21. Expires 130
16.22. From 130
16.23. If-Match 131
16.24. If-Modified-Since 131
16.25. If-None-Match 132
16.26. Last-Modified 132
16.27. Location 133
16.28. Media-Properties 133
16.29. Media-Range 135
16.30. MTag 136
16.31. Notify-Reason 136
16.32. Pipelined-Requests 136
16.33. Proxy-Authenticate 137
16.34. Proxy-Authorization 138
16.35. Proxy-Require 138
16.36. Proxy-Supported 139
16.37. Public 139
16.38. Range 140
16.39. Referrer 142
16.40. Request-Status 142
16.41. Require 143
16.42. Retry-After 144
16.43. RTP-Info 144
16.44. Scale 146
16.45. Seek-Style 147
16.46. Server 149
16.47. Session 149
16.48. Speed 150
16.49. Supported 151
16.50. Terminate-Reason 152
16.51. Timestamp 152
16.52. Transport 153
16.53. Unsupported 160
16.54. User-Agent 160
16.55. Vary 160

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1299
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.56. Via 161
16.57. WWW-Authenticate 162
17. Proxies 163
17.1. Proxies and Protocol Extensions 164
17.2. Multiplexing and Demultiplexing of Messages 165
18. Caching 166
18.1. Validation Model 166
18.1.1. Last-Modified Dates 168
18.1.2. Message Body Tag Cache Validators 168
18.1.3. Weak and Strong Validators 168
18.1.4. Rules for When to Use Message Body Tags and Last-Modified Dates 170
18.1.5. Non-validating Conditionals 172
18.2. Invalidation After Updates or Deletions 172
19. Security Framework 174
19.1. RTSP and HTTP Authentication 174
19.2. RTSP over TLS 174
19.3. Security and Proxies 175
19.3.1. Accept-Credentials 176
19.3.2. User approved TLS procedure 177
20. Syntax 180
20.1. Base Syntax 180
20.2. RTSP Protocol Definition 182
20.2.1. Generic Protocol elements 182
20.2.2. Message Syntax 185
20.2.3. Header Syntax 189
20.3. SDP extension Syntax 198
21. Security Considerations 199
21.1. Remote denial of Service Attack 201
22. IANA Considerations 203
22.1. Feature-tags 203
22.1.1. Description 203
22.1.2. Registering New Feature-tags with IANA 204
22.1.3. Registered entries 204
22.2. RTSP Methods 204
22.2.1. Description 204
22.2.2. Registering New Methods with IANA 205
22.2.3. Registered Entries 205
22.3. RTSP Status Codes 205
22.3.1. Description 205
22.3.2. Registering New Status Codes with IANA 206
22.3.3. Registered Entries 206
22.4. RTSP Headers 206
22.4.1. Description 206
22.4.2. Registering New Headers with IANA 206
22.4.3. Registered entries 207
22.5. Accept-Credentials 207
22.5.1. Accept-Credentials policies 207

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1300
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.5.2. Accept-Credentials hash algorithms 208
22.6. Cache-Control Cache Directive Extensions 208
22.7. Media Properties 209
22.7.1. Description 209
22.7.2. Registration Rules 209
22.7.3. Registered Values 210
22.8. Notify-Reason header 210
22.8.1. Description 210
22.8.2. Registration Rules 210
22.8.3. Registered Values 211
22.9. Range header formats 211
22.9.1. Description 211
22.9.2. Registration Rules 211
22.9.3. Registered Values 211
22.10. Terminate-Reason Header 212
22.10.1. Redirect Reasons 212
22.10.2. Terminate-Reason Header Parameters 212
22.11. RTP-Info header parameters 212
22.11.1. Description 212
22.11.2. Registration Rules 213
22.11.3. Registered Values 213
22.12. Seek-Style Policies 213
22.12.1. Description 213
22.12.2. Registration Rules 213
22.12.3. Registered Values 214
22.13. Transport Header Registries 214
22.13.1. Transport Protocol Specification 214
22.13.2. Transport modes 216
22.13.3. Transport Parameters 216
22.14. URI Schemes 216
22.14.1. The rtsp URI Scheme 216
22.14.2. The rtsps URI Scheme 217
22.14.3. The rtspu URI Scheme 218
22.15. SDP attributes 219
22.16. Media Type Registration for text/parameters 220
23. References 222
23.1. Normative References 222
23.2. Informative References 224
Appendix A. Examples 227
A.1. Media on Demand (Unicast) 227
A.2. Media on Demand using Pipelining 231
A.3. Media on Demand (Unicast) 233
A.4. Single Stream Container Files 237
A.5. Live Media Presentation Using Multicast 239
A.6. Capability Negotiation 240
Appendix B. RTSP Protocol State Machine 242
B.1. States 242
B.2. State variables 242

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1301
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

B.3. Abbreviations 242
B.4. State Tables 243
Appendix C. Media Transport Alternatives 249
C.1. RTP 249
C.1.1. AVP 249
C.1.2. AVP/UDP 249
C.1.3. AVPF/UDP 250
C.1.4. SAVP/UDP 251
C.1.5. SAVPF/UDP 253
C.1.6. RTCP usage with RTSP 253
C.2. RTP over TCP 255
C.2.1. Interleaved RTP over TCP 255
C.2.2. RTP over independent TCP 255
C.3. Handling Media Clock Time Jumps in the RTP Media Layer . 259
C.4. Handling RTP Timestamps after PAUSE 263
C.5. RTSP / RTP Integration 265
C.6. Scaling with RTP 265
C.7. Maintaining NPT synchronization with RTP timestamps . . 265
C.8. Continuous Audio 265
C.9. Multiple Sources in an RTP Session 265
C.10. Usage of SSRCs and the RTCP BYE Message During an RTSP Session 265
C.11. Future Additions 266
Appendix D. Use of SDP for RTSP Session Descriptions 267
D.1. Definitions 267
D.1.1. Control URI 267
D.1.2. Media Streams 268
D.1.3. Payload Type(s) 269
D.1.4. Format-Specific Parameters 269
D.1.5. Directionality of media stream 269
D.1.6. Range of Presentation 270
D.1.7. Time of Availability 271
D.1.8. Connection Information 271
D.1.9. Message Body Tag 271
D.2. Aggregate Control Not Available 272
D.3. Aggregate Control Available 272
D.4. Grouping of Media Lines in SDP 273
D.5. RTSP external SDP delivery 274
Appendix E. RTSP Use Cases 275
E.1. On-demand Playback of Stored Content 275
E.2. Unicast Distribution of Live Content 276
E.3. On-demand Playback using Multicast 277
E.4. Inviting an RTSP server into a conference 277
E.5. Live Content using Multicast 278
Appendix F. Text format for Parameters 280
Appendix G. Requirements for Unreliable Transport of RTSP . . . 281
Appendix H. Backwards Compatibility Considerations 283
H.1. Play Request in Play State 283

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1302
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

H.2. Using Persistent Connections 283
Appendix I. Open Issues 284
Appendix J. Changes 285
J.1. Brief Overview 285
J.2. Detailed List of Changes 286
Appendix K. Acknowledgements 293
K.1. Contributors 293
Appendix L. RFC Editor Consideration 295
Authors' Addresses 296

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1303
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

1. Introduction

This memo defines version 2.0 of the Real Time Streaming Protocol (RTSP 2.0). RTSP 2.0 is an application-level protocol for setup and control over the delivery of data with real-time properties, typically streaming media. Streaming media is, for instance, video on demand or audio live streaming. Put simply, RTSP acts as a "network remote control" for multimedia servers, similar to the remote control for a DVD player.

The protocol operates between RTSP 2.0 clients and servers, but also supports the usage of proxies placed between clients and servers. Clients can request information about streaming media from servers by asking for a description of the media or use media description provided externally. The media delivery protocol is used to establish the media streams described by the media description. Clients can then request to play out the media, pause it, or stop it completely, as known from DVD players remote control or media players. The requested media can consist of multiple audio and video streams that are delivered as a time-synchronized streams from servers to clients.

RTSP 2.0 is a replacement of RTSP 1.0 [RFC2326] that obsoletes that specification. This protocol is based on RTSP 1.0 but is not backwards compatible other than in the basic version negotiation mechanism. The changes are documented in Appendix J. There are many

reasons why RTSP 2.0 can't be backwards compatible with RTSP 1.0 but some of the main ones are:

- Most headers that needed to be extensible did not define the allowed syntax, preventing safe deployment of extensions;
- The changed behavior of the PLAY method when received in Play state;
- Changed behavior of the extensibility model and its mechanism;
- The change of syntax for some headers.

In summary, there are so many small details that changing version become necessary to enable clarification and consistent behavior.

This document is structured as follows. It begins with an overview of the protocol operations and its functions in an informal way. Then a set of definitions of used terms and document conventions is introduced. It is followed by the actual protocol specification. In the appendix some functionality that isn't core RTSP, but still

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1304
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

important to enable some usage, is defined. RTP usage is defined in Appendix C and SDP usage with RTSP Appendix D, making these two appendixes mandatory. This is followed by a number of informational parts discussing the changes, use cases, different considerations or motivations.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1305
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

2. Protocol Overview

This section provides a informative overview of the different mechanisms in the RTSP 2.0 protocol, to give the reader a high level understanding before getting into all the different details. In case of conflict with this description and the later sections, the later sections take precedence. For more information about considered use cases for RTSP see Appendix E.

RTSP 2.0 is a bi-directional request and response protocol that first establishes a context including content resources (the media) and then controls the delivery of these content resources from the provider to the consumer. RTSP has three fundamental parts: Session Establishment, Media Delivery Control, and an extensibility model described below. The protocol is based on some assumptions about existing functionality to provide a complete solution for client controlled real-time media delivery.

RTSP uses text-based messages, requests and responses, that may contain a binary message body. An RTSP request starts with a method line that identifies the method, the protocol and version and the resource to act on. Following the method line are a number of RTSP headers. This part is ended by two consecutive carriage return line feed (CRLF) character pairs. The message body if present follows the two CRLF and the body's length are described by a message header. RTSP responses are similar, but start with a response line with the protocol and version, followed by a status code and a reason phrase. RTSP messages are sent over a reliable transport protocol between the client and server. RTSP 2.0 requires clients and servers to implement TCP, and TLS over TCP, as mandatory transports for RTSP messages.

2.1. Presentation Description

RTSP exists to provide access to multi-media presentations and content, but tries to be agnostic about the media type or the actual media delivery protocol that is used. To enable a client to implement a complete system, an RTSP-external mechanism for describing the presentation and the delivery protocol(s) is used. RTSP assumes that this description is either delivered completely out of bands or as a data object in the response to a client's request using the DESCRIBE method (Section 13.2).

Parameters that commonly have to be included in the Content Description are the following:

- Number of media streams

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1306
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- The resource identifier for each media stream/resource that is to be controlled by RTSP
- The protocol that each media stream is to be delivered over
- Transport protocol parameters that are not negotiated or vary with each client
- Media encoding information enabling a client to correctly decode it upon reception
- An aggregate control resource identifier

RTSP uses its own URI schemes ("rtsp" and "rtsps") to reference media resources and aggregates under common control.

This specification describes in Appendix D how one uses SDP [RFC4566] for Content Description

2.2. Session Establishment

The RTSP client can request the establishment of an RTSP session after having used the presentation description to determine which media streams are available, and also which media delivery protocol is used and their particular resource identifiers. The RTSP session is a common context between the client and the server that consist of one or more media resources that are to be under common media delivery control.

The client creates an RTSP session by sending a request using the SETUP method (Section 13.3) to the server. In the SETUP request the client also includes all the transport parameters necessary to enable the media delivery protocol to function in the "Transport" header (Section 16.52). This includes parameters that are pre-established by the presentation description but necessary for any middlebox to correctly handle the media delivery protocols. The Transport header in a request may contain multiple alternatives for media delivery in a prioritized list, which the server can select from. These alternatives are typically based on information in the content description.

The server determines if the media resource is available upon receiving a SETUP request and if any of the transport parameter specifications are acceptable. If that is successful, an RTSP session context is created and the relevant parameters and state is stored. An identifier is created for the RTSP session and included in the response in the Session header (Section 16.47). The SETUP response includes a Transport header that specifies which of the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1307
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

alternatives has been selected and relevant parameters.

A SETUP request that references an existing RTSP session but identifies a new media resource is a request to add that media resource under common control with the already present media resources in an aggregated session. A client can expect this to work for all media resources under RTSP control within a multi-media content. However, aggregating resources from different content are likely to be refused by the server. The RTSP session as aggregate is referenced by the aggregate control URI, even if the RTSP session only contains a single media.

To avoid an extra round trip in the session establishment of aggregated RTSP sessions, RTSP 2.0 supports pipelined requests; i.e., the client can send multiple requests back-to-back without waiting first for the completion of any of them. The client uses client-selected identifier in the Pipelined-Requests header to instruct the server to bind multiple requests together as if they included the session identifier.

The SETUP response also provides additional information about the established sessions in a couple of different headers. The Media-Properties header includes a number of properties that apply for the aggregate that is valuable when doing media delivery control and configuring user interface. The Accept-Ranges header informs the client about which range formats that the server supports with these media resources. The Media-Range header inform the client about the time range of the media currently available.

2.3. Media Delivery Control

After having established an RTSP session, the client can start controlling the media delivery. The basic operations are Start by using the PLAY method (Section 13.4) and Halt by using the PAUSE method (Section 13.6). PLAY also allows for choosing the starting media position from which the server should deliver the media. The positioning is done using the Range header (Section 16.38) that supports several different time formats: Normal Play Time (Section 4.5), SMPTE Timestamps (Section 4.4) and absolute time (Section 4.6). The Range header does further allow the client to specify a position where delivery should end, thus allowing a specific interval to be delivered.

The support for positioning/searching within a content depends on the content's media properties. Content exists in a number of different types, such as: on-demand, live, and live with simultaneous recording. Even within these categories there are differences in how the content is generated and distributed, which affect how it can be

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1308
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

accessed for playback. The properties applicable for the RTSP session are provided by the server in the SETUP response using the Media-Properties header (Section 16.28). These are expressed using one or several independent attributes. A first attribute is Random Access, which expresses if positioning can be done, and with what granularity. Another aspect is whether the content will change during the lifetime of the session. While on-demand content will provided in full from the beginning, a live stream being recorded results in the length of the accessible content growing as the session goes on. There also exist content that is dynamically built by another protocol than RTSP and thus also changes in steps during the session, but maybe not continuously. Furthermore, when content is recorded, there are cases where not the complete content is maintained, but, for example, only the last hour. All these properties result in the need for mechanisms that will be discussed below.

When the client accesses on-demand content that allows random access in, the client can issue the PLAY request for any point in the content between the start and the end. The server will deliver media from the closest random access point prior to the requested point and indicate that in its PLAY response. If the client issues a PAUSE, the delivery will be halted and the point at which the server stopped will be reported back in the response. The client can later resume by a sending PLAY request without a range header. When the server is about to complete the PLAY request by delivering the end of the content or the requested range, the server will send a PLAY_NOTIFY request indicating this.

When playing live content with no extra functions, such as recording, the client will receive the live media from the server after having sent a PLAY request. Seeking in such content is not possible as the server does not store it, but only forwards it from the source of the session. Thus delivery continues until the client sends a PAUSE request, tears down the session, or the content ends.

For live sessions that are being recorded the client will need to keep track of how the recording progresses. Upon session establishment the client will learn the current duration of the recording from the Media-Range header. As the recording is ongoing the content grows in direct relation to the passed time. Therefore, each server's response to a PLAY request will contain the current Media-Range header. The server should also regularly send every 5 minutes the current media range in a PLAY_NOTIFY request. If the live transmission ends, the server must send a PLAY_NOTIFY request with the updated Media-Properties indicating that the content stopped being a recorded live session and instead become a on-demand content; the request also contains the final media range. While the live

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1309
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

delivery continues the client can request to play the current live point by using the NPT timescale symbol "now", or it can request a specific point in the available content by an explicit range request for that point. If the requested point is outside of the available interval the server will adjust the position to the closest available point, i.e., either at the beginning or the end.

A special case of recording is that where the recording is not retained longer than a specific time period, thus as the live delivery continues the client can access any media within a moving window that covers, for example, "now" to "now" minus 1 hour. A client that pauses on a specific point within the content may not be able to retrieve the content anymore. If the client waits too long before resuming the pause point, the content may no longer be available. In this case the pause point will be adjusted to the end of the available media.

2.4. Session Parameter Manipulations

A session may have additional state or functionality that effects how the server or client treats the session, content, how it functions, or feedback on how well the session works. Such extensions are not defined in this specification, but may be done in various extensions. RTSP has two methods for retrieving and setting parameter values on either the client or the server: GET_PARAMETER (Section 13.8) and SET_PARAMETER (Section 13.9). These methods carry the parameters in a message body of the appropriate format. One can also use headers to query state with the GET_PARAMETER method. As an example, clients needing to know the current media-range for a time-progressing session can use the GET_PARAMETER method and include the media-range. Furthermore, synchronization information can be requested by using a combination of RTP-Info and Range.

RTSP 2.0 does not have a strong mechanism for providing negotiation of which headers, or parameters and their formats, that can be used. However, responses will indicate request headers or parameters that are not supported. A priori determination of what features are available needs to be done through out-of-band mechanisms, like the session description, or through the usage of feature tags (Section 4.7).

2.5. Media Delivery

The delivery of media to the RTSP client is done with a protocol outside of RTSP and this protocol is determined during the session establishment. This document specifies how media is delivered with RTP over UDP, TCP or the RTSP control connection. Additional protocols may be specified in the future based on demand.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1310
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The usage of RTP as media delivery protocol requires some additional information to function well. The PLAY response contains information to enable reliable and timely deliver of how a client should synchronize different sources in the different RTP sessions. It also provides a mapping between RTP timestamps and the content time scale. When the server want to notify the client about the completion of the media delivery, it sends a PLAY_NOTIFY request to the client. The PLAY_NOTIFY request includes information about the stream end, including the last RTP sequence number for each stream, thus enabling the client to empty the buffer smoothly.

2.5.1. Media Delivery Manipulations

The basic playback functionality of RTSP enables delivery of a range of requested content to the client at the pace intended by the content's creator. However, RTSP can also manipulate the delivery to the client in two ways.

Scale: The ratio of media content time delivered per unit playback time.

Speed: The ratio of playback time delivered per unit of wallclock time.

Both affect the media delivery per time unit. However, they manipulate two independent time scales and the effects are possible to combine.

Scale is used for fast forward or slow motion control as it changes the amount of content timescale that should be played back per time unit. Scale > 1.0, means fast forward, e.g. Scale=2.0 results in that 2 seconds of content is played back every second of playback. Scale = 1.0 is the default value that is used if no Scale is specified, i.e., playback at the content's original rate. Scale values between 0 and 1.0 is providing for slow motion. Scale can be negative to allow for reverse playback in either regular pace (Scale = -1.0) or fast backwards (Scale < -1.0) or slow motion backwards (-1.0 < Scale < 0). Scale = 0 is equal to pause and is not allowed.

In most cases the realization of scale means server side manipulation of the media to ensure that the client can actually play it back. These media manipulation and when they are needed are highly media-type dependent. Lets exemplify with two common media types audio and video.

It is very difficult to modify the playback rate of audio. A maximum of 10-30% is possible by changing the pitch-rate of speech. Music goes out of tune if one tries to manipulate the playback rate by

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1311
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

resampling it. This is a well known problem and audio is commonly muted or played back in short segments with skips to keep up with the current playback point.

For video is possible to manipulate the frame rate, although the rendering capabilities are often limited to certain frame rates. Also the allowed bitrates in decoding, the structured used in the encoding and the dependency between frames and other capabilities of the rendering device limits the possible manipulations. Therefore, the basic fast forward capabilities often are implemented by selecting certain subsets of frames.

Due to the media restrictions, the possible scale values are commonly restricted to the set of realizable scale ratios. To enable the clients to select from the possible scale values, RTSP can signal the supported Scale ratios for the content. To support aggregated or dynamic content, where this may change during the ongoing session and dependent on the location within the content, a mechanism for updating the media properties and the currently used scale factor exist.

Speed affects how much of the playback timeline is delivered in a given wallclock period. The default is Speed = 1 which means to deliver at the same rate the media is consumed. Speed > 1 means that the receiver will get content faster than it regularly would consume it. Speed < 1 means that delivery is slower than the regular media rate. Speed values of 0 or lower have no meaning and are not allowed. This mechanism enables two general functionalities. One is client side scale operations, i.e. the client receives all the frames and makes the adjustment to the playback locally. The second is delivery control for buffering of media. By specifying a speed over 1.0 the client can build up the amount of playback time it has present in its buffers to a level that is sufficient for its needs.

A naive implementation of Speed would only affect the transmission schedule of the media and has a clear impact on the needed bandwidth. This would result in the data rate being proportional to the speed factor. Speed = 1.5, i.e., 50% faster than normal delivery, would result in a 50% increase in the data transport rate. If that can be supported or not depends solely on the underlying network path. Scale may also have some impact on the required bandwidth due to the manipulation of the content in the new playback schedule. An example is fast forward where only the independently decodable intra frames are included in the media stream. This usage of solely intra frames increases the data rate significantly compared to a normal sequence with the same number of frames, where most frames are encoded using prediction.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1312
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This potential increase of the data rate needs to be handled by the media sender. The client has requested that the media will be delivered in a specific way, which should be honored. However, the media sender cannot ignore if the network path between the sender and the receiver can't handle the resulting media stream. In that case the media stream needs to be adapted to fit the available resources of the path. This can result in a reduced media quality.

The need for bitrate adaptation becomes especially problematic in connection with the Speed semantics. If the goal is to fill up the buffer, the client may not want to do that at the cost of reduced quality. If the client wants to make local playout changes then it may actually require that the requested speed be honored. To resolve this issue, Speed uses a range so that both cases can be supported. The server is requested to use the highest possible speed value within the range which is compatible with the available bandwidth. As long as the server can maintain a speed value within the range it shall not change the media quality, but instead modify the actual delivery rate in response to available bandwidth and reflect this in the Speed value in the response. However, if this is not possible, the server should instead modify the media quality to respect the lowest speed value and the available bandwidth.

This functionality enables the local scaling implementation to use a tight range, or even a range where the lower bound equals the upper bound, to identify that it requires the server to deliver the requested amount of media time per delivery time independent of how much it needs to adapt the media quality to fit within the available path bandwidth. For buffer filling, it is suitable to use a range with a reasonable span and with a lower bound at the nominal media rate 1.0, such as 1.0 - 2.5. If the client wants to reduce the buffer, it can specify an upper bound that is below 1.0 to force the server to deliver slower than the nominal media rate.

2.6. Session Maintenance and Termination

The session context that has been established is kept alive by having the client show liveness. This is done in two main ways:

- Media transport protocol keep-alive. RTCP may be used when using RTP.
- Any RTSP request referencing the session context.

Section 10.5 discusses the methods for showing liveness in more depth. If the client fails to show liveness for more than the established session timeout value (normally 60 seconds), the server may terminate the context. Other values may be selected by the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1313
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

server through the inclusion of the timeout parameter in the session header.

The session context is normally terminated by the client sending a TEARDOWN request to the server referencing the aggregated control URI. An individual media resource can be removed from a session context by a TEARDOWN request referencing that particular media resource. If all media resources are removed from a session context, the session context is terminated.

A client may keep the session alive indefinitely if allowed by the server; however, it is recommended to release the session context when an extended period of time without media delivery activity has passed. The client can re-establish the session context if required later. What constitutes an extended period of time is dependent on the server and its usage. It is recommended that the client terminates the session before 10*times the session timeout value has passed. A server may terminate the session after one session timeout period without any client activity beyond keep-alive. When a server terminates the session context, it does that by sending a TEARDOWN request indicating the reason.

A server can also request that the client tear down the session and re-establish it at an alternative server, as may be needed for maintenance. This is done by using the REDIRECT method. The Terminate-Reason header is used to indicate when and why. The Location header indicates where it should connect if there is an alternative server available. When the deadline expires, the server simply stops providing the service. To achieve a clean closure, the client needs to initiate session termination prior to the deadline. In case the server has no other server to redirect to, and wants to close the session for maintenance, it shall use the TEARDOWN method with a Terminate-Reason header.

2.7. Extending RTSP

RTSP is quite a versatile protocol which supports extensions in many different directions. Even this core specification contains several blocks of functionality that are optional to implement. The use case and need for the protocol deployment should determine what parts are implemented. Allowing for extensions makes it possible for RTSP to reach out to additional use cases. However, extensions will affect the interoperability of the protocol and therefore it is important that they can be added in a structured way.

The client can learn the capability of a server by using the OPTIONS method (Section 13.1) and the Supported header (Section 16.49). It can also try and possibly fail using new methods, or require that

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1314
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

particular features are supported using the Require or Proxy-Require header.

The RTSP protocol in itself can be extended in three ways, listed here in order of the magnitude of changes supported:

- Existing methods can be extended with new parameters, for example, headers, as long as these parameters can be safely ignored by the recipient. If the client needs negative acknowledgment when a method extension is not supported, a tag corresponding to the extension may be added in the field of the Require or Proxy-Require headers (see Section 16.35).

- New methods can be added. If the recipient of the message does not understand the request, it must respond with error code 501 (Not Implemented) so that the sender can avoid using this method again. A client may also use the OPTIONS method to inquire about methods supported by the server. The server must list the methods it supports using the Public response header.

- A new version of the protocol can be defined, allowing almost all aspects (except the position of the protocol version number) to change. A new version of the protocol must be registered through an IETF standard track document.

The basic capability discovery mechanism can be used to both discover support for a certain feature and to ensure that a feature is available when performing a request. For a detailed explanation of this see Section 11.

New media delivery protocols may be added and negotiated at session establishment, in addition to extension to the core protocol. Certain types of protocol manipulations can be done through parameter formats using SET_PARAMETER and GET_PARAMETER.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1315
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

3. Document Conventions

3.1. Notational Conventions

Since a few of the definitions are identical to HTTP/1.1, this specification only points to the section where they are defined rather than copying it. For brevity, [HX.Y] is to be taken to refer to Section X.Y of the current HTTP/1.1 specification ([RFC2616]).

All the mechanisms specified in this document are described in both prose and the Augmented Backus-Naur form (ABNF) described in detail in [RFC5234].

Indented and smaller-type paragraphs are used to provide informative background and motivation. This is intended to give readers who were not involved with the formulation of the specification an understanding of why things are the way they are in RTSP.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The word, "unspecified" is used to indicate functionality or features that are not defined in this specification. Such functionality cannot be used in a standardized manner without further definition in an extension specification to RTSP.

3.2. Terminology

Aggregate control: The concept of controlling multiple streams using a single timeline, generally maintained by the server. A client, for example, uses aggregate control when it issues a single play or pause message to simultaneously control both the audio and video in a movie. A session which is under aggregate control is referred to as an aggregated session.

Aggregate control URI: The URI used in an RTSP request to refer to and control an aggregated session. It normally, but not always, corresponds to the presentation URI specified in the session description. See Section 13.3 for more information.

Client: The client requests media service from the media server.

Connection: A transport layer virtual circuit established between two programs for the purpose of communication.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1316
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Container file: A file which may contain multiple media streams which often constitutes a presentation when played together. The concept of a container file is not embedded in the protocol. However, RTSP servers may offer aggregate control on the media streams within these files.

Continuous media: Data where there is a timing relationship between source and sink; that is, the sink needs to reproduce the timing relationship that existed at the source. The most common examples of continuous media are audio and motion video. Continuous media can be real-time (interactive or conversational), where there is a "tight" timing relationship between source and sink, or streaming where the relationship is less strict.

Feature-tag: A tag representing a certain set of functionality, i.e. a feature.

IRI: Internationalized Resource Identifier, is the same as an URI, with the exception that it allows characters from the whole Universal Character Set (Unicode/ISO 10646), rather than the US-ASCII only. See [RFC3987] for more information.

Live: Normally used to describe a presentation or session with media coming from an ongoing event. This generally results in the session having an unbound or only loosely defined duration, and sometimes no seek operations are possible.

Media initialization: Datatype/codec specific initialization. This includes such things as clock rates, color tables, etc. Any transport-independent information which is required by a client for playback of a media stream occurs in the media initialization phase of stream setup.

Media parameter: Parameter specific to a media type that may be changed before or during stream delivery.

Media server: The server providing media delivery services for one or more media streams. Different media streams within a presentation may originate from different media servers. A media server may reside on the same host or on a different host from which the presentation is invoked.

(Media) stream: A single media instance, e.g., an audio stream or a video stream as well as a single whiteboard or shared application group. When using RTP, a stream consists of all RTP and RTCP packets created by a source within an RTP session.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1317
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Message: The basic unit of RTSP communication, consisting of a structured sequence of octets matching the syntax defined in Section 20 and transmitted over a connection or a connectionless transport. A message is either a Request or a Response.

Message Body: The information transferred as the payload of a message (Request and response). A message body consists of meta-information in the form of message-body headers and content in the form of a message-body, as described in Section 9.

Non-Aggregated Control: Control of a single media stream.

Presentation: A set of one or more streams presented to the client as a complete media feed and described by a presentation description as defined below. Presentations with more than one media stream are often handled in RTSP under aggregate control.

Presentation description: A presentation description contains information about one or more media streams within a presentation, such as the set of encodings, network addresses and information about the content. Other IETF protocols such as SDP ([RFC4566]) use the term "session" for a presentation. The presentation description may take several different formats, including but not limited to the session description protocol format, SDP.

Response: An RTSP response to a Request. One type of RTSP message. If an HTTP response is meant, it is indicated explicitly.

Request: An RTSP request. One type of RTSP message. If an HTTP request is meant, it is indicated explicitly.

Request-URI: The URI used in a request to indicate the resource on which the request is to be performed.

RTSP agent: Refers to either an RTSP client, an RTSP server, or an RTSP proxy. In this specification, there are many capabilities that are common to these three entities such as the capability to send requests or receive responses. This term will be used when describing functionality that is applicable to all three of these entities.

RTSP session: A stateful abstraction upon which the main control methods of RTSP operate. An RTSP session is a common context; it is created, maintained and destroyed on client's request. It is established by an RTSP server upon the completion of a successful SETUP request (when a 200 OK response is sent) and is labeled with a session identifier at that time. The session exists until timed out by the server or explicitly removed by a TEARDOWN request. An

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1318
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RTSP session is a stateful entity; an RTSP server maintains an explicit session state machine (see Appendix B) where most state transitions are triggered by client requests. The existence of a session implies the existence of state about the session's media streams and their respective transport mechanisms. A given session can have one or more media streams associated with it. An RTSP server uses the session to aggregate control over multiple media streams.

Origin Server: The server on which a given resource resides.

Transport initialization: The negotiation of transport information (e.g., port numbers, transport protocols) between the client and the server.

URI: Universal Resource Identifier, see [RFC3986]. The URIs used in RTSP are generally URLs as they give a location for the resource. As URLs are a subset of URIs, they will be referred to as URIs to cover also the cases when an RTSP URI would not be an URL.

URL: Universal Resource Locator, is an URI which identifies the resource through its primary access mechanism, rather than identifying the resource by name or by some other attribute(s) of that resource.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1319
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4. Protocol Parameters

4.1. RTSP Version

This specification defines version 2.0 of RTSP.

RTSP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further RTSP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values.

The <minor> number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The <major> number is incremented when the format of a message within the protocol is changed. The version of an RTSP message is indicated by an RTSP-Version field in the first line of the message. Note that the major and minor numbers MUST be treated as separate integers and that each MAY be incremented higher than a single digit. Thus, RTSP/2.4 is a lower version than RTSP/2.13, which in turn is lower than RTSP/12.3. Leading zeros MUST be ignored by recipients and MUST NOT be sent.

4.2. RTSP IRI and URI

RTSP 2.0 defines and registers three URI schemes "rtsp", "rtsps" and "rtspu". The usage of the last, "rtspu", is unspecified in RTSP 2.0, and is defined here to register and reserve the URI scheme that is defined in RTSP 1.0. The "rtspu" scheme indicates unspecified transport of the RTSP messages over unreliable transport (UDP in RTSP 1.0). A RTSP server MUST response with an error code indicating the "rtspu" scheme is not implemented (501) to a request that carries a "rtspu" URI scheme. The details of the syntax of "rtsp" and "rtsps" URIs has been changed from RTSP 1.0.

This specification also defines the format of the RTSP IRI [RFC3987] that can be used as RTSP resource identifiers and locators, in web pages, user interfaces, on paper, etc. However, the RTSP request message format only allows usage of the absolute URI format. The RTSP IRI format MUST use the rules and transformation for IRIs defined in [RFC3987]. This way RTSP 2.0 URIs for request can be produced from an RTSP IRI.

The RTSP IRI and URI are both syntax restricted compared to the generic syntax defined in [RFC3986] and [RFC3987]:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1320
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- An absolute URI requires the authority part; i.e., a host identity must be provided.

- Parameters in the path element are prefixed with the reserved separator ";".

The RTSP URI and IRI is case sensitive, with the exception of those parts that [RFC3986] and [RFC3987] defines as case-insensitive; for example, the scheme and host part.

The fragment identifier is used as defined in sections 3.5 and 4.3 of [RFC3986], i.e. the fragment is to be stripped from the IRI by the requester and not included in the request URI. The user agent needs to interpret the value of the fragment based on the media type the request relates to; i.e., the media type indicated in Content-Type header in the response to DESCRIBE.

The syntax of any URI query string is unspecified and responder (usually the server) specific. The query is, from the requester's perspective, an opaque string and needs to be handled as such. Please note that relative URI with queries are difficult to handle due to the RFC 3986 relative URI handling rules. Any change of the path element using a relative URI results in the stripping of the query, which means the relative part needs to contain the query.

The URI scheme "rtsp" requires that commands are issued via a reliable protocol (within the Internet, TCP), while the scheme "rtsps" identifies a reliable transport using secure transport (TLS [RFC5246], see (Section 19).

For the scheme "rtsp", if no port number is provided in the authority part of the URI port number 554 MUST be used. For the scheme "rtsps", the TCP port 322 is registered and MUST be assumed.

A presentation or a stream is identified by a textual media identifier, using the character set and escape conventions of URIs [RFC3986]. URIs may refer to a stream or an aggregate of streams; i.e., a presentation. Accordingly, requests described in (Section 13) can apply to either the whole presentation or an individual stream within the presentation. Note that some request methods can only be applied to streams, not presentations, and vice versa.

For example, the RTSP URI:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1321
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

rtsp://media.example.com:554/twister/audiotrack

may identify the audio stream within the presentation "twister", which can be controlled via RTSP requests issued over a TCP connection to port 554 of host media.example.com.

Also, the RTSP URI:

rtsp://media.example.com:554/twister

identifies the presentation "twister", which may be composed of audio and video streams, but could also be something else like a random media redirector.

This does not imply a standard way to reference streams in URIs. The presentation description defines the hierarchical relationships in the presentation and the URIs for the individual streams. A presentation description may name a stream "a.mov" and the whole presentation "b.mov".

The path components of the RTSP URI are opaque to the client and do not imply any particular file system structure for the server.

This decoupling also allows presentation descriptions to be used with non-RTSP media control protocols simply by replacing the scheme in the URI.

4.3. Session Identifiers

Session identifiers are strings of length 8-128 characters. A session identifier MUST be chosen cryptographically random (see [RFC4086]) . It is RECOMMENDED that it contains 128 bits of entropy, i.e. approximately 22 characters from a high quality generator. (see Section 21.) However, note that the session identifier does not provide any security against session hijacking unless it is kept confidential by the client, server and trusted proxies.

4.4. SMPTE Relative Timestamps

A SMPTE relative timestamp expresses time relative to the start of the clip. Relative timestamps are expressed as SMPTE time codes for frame-level access accuracy. The time code has the format

hours:minutes:seconds:frames.subframes,

with the origin at the start of the clip. The default SMPTE format is "SMPTE 30 drop" format, with frame rate is 29.97 frames per second. Other SMPTE codes MAY be supported (such as "SMPTE 25")

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1322
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

through the use of "smpte-type". For SMPTE 30, the "frames" field in the time value can assume the values 0 through 29. The difference between 30 and 29.97 frames per second is handled by dropping the first two frame indices (values 00 and 01) of every minute, except every tenth minute. If the frame and the subframe values are zero, they may be omitted. Subframes are measured in one-hundredth of a frame.

Examples:

```
smpte=10:12:33:20-
smpte=10:07:33-
smpte=10:07:00-10:07:33:05.01
smpte-25=10:07:00-10:07:33:05.01
```

4.5. Normal Play Time

Normal play time (NPT) indicates the stream absolute position relative to the beginning of the presentation, not to be confused with the Network Time Protocol (NTP) [RFC5905]. The timestamp consists of two parts: the mandatory first part may be expressed in either seconds or hours, minutes, and seconds. The optional second part consists of a decimal point and decimal figures and indicates fractions of a second.

The beginning of a presentation corresponds to 0.0 seconds. Negative values are not defined.

The special constant "now" is defined as the current instant of a live event. It MAY only be used for live events, and MUST NOT be used for on-demand (i.e., non-live) content.

NPT is defined as in DSM-CC [ISO.13818-6.1995]: "Intuitively, NPT is the clock the viewer associates with a program. It is often digitally displayed on a VCR. NPT advances normally when in normal play mode (scale = 1), advances at a faster rate when in fast scan forward (high positive scale ratio), decrements when in scan reverse (negative scale ratio) and is fixed in pause mode. NPT is (logically) equivalent to SMPTE time codes."

Examples:

```
npt=123.45-125
npt=12:05:35.3-
npt=now-
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1323
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The syntax conforms to ISO 8601 [ISO.8601.2000]. The npt-sec notation is optimized for automatic generation, the npt-hhmmss notation for consumption by human readers. The "now" constant allows clients to request to receive the live feed rather than the stored or time-delayed version. This is needed since neither absolute time nor zero time are appropriate for this case.

4.6. Absolute Time

Absolute time is expressed as ISO 8601 [ISO.8601.2000] timestamps, using UTC (GMT). Fractions of a second may be indicated.

Example for November 8, 1996 at 14h 37 min and 20 and a quarter seconds UTC:

```
19961108T143720.25Z
```

4.7. Feature-Tags

Feature-tags are unique identifiers used to designate features in RTSP. These tags are used in Require (Section 16.41), Proxy-Require (Section 16.35), Proxy-Supported (Section 16.36), and Unsupported (Section 16.53) header fields.

A feature-tag definition MUST indicate which combination of clients, servers or proxies they applies to.

The creator of a new RTSP feature-tag should either prefix the feature-tag with a reverse domain name (e.g., "com.example.mynewfeature" is an apt name for a feature whose inventor can be reached at "example.com"), or register the new feature-tag with the Internet Assigned Numbers Authority (IANA) (see IANA Section 22).

The usage of feature-tags is further described in Section 11 that deals with capability handling.

4.8. Message Body Tags

Message body tags are opaque strings that are used to compare two message bodies from the same resource, for example in caches or to optimize setup after a redirect. Message body tags can be carried in the MTag header (see Section 16.30) or in SDP (see Appendix D.1.9). MTag is similar to ETag in HTTP/1.1.

A message body tag MUST be unique across all versions of all message bodies associated with a particular resource. A given message body tag value MAY be used for message bodies obtained by requests on

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1324
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

different URIs. The use of the same message body tag value in conjunction with message bodies obtained by requests on different URIs does not imply the equivalence of those message bodies

Message body tags are used in RTSP to make some methods conditional. The methods are made conditional through the inclusion of headers; see "If-Match" (Section 16.23) and "If-None-Match" (Section 16.25). Note that RTSP message body tags apply to the complete presentation; i.e., both the presentation description and the individual media streams. Thus message body tags can be used to verify at setup time after a redirect that the same session description applies to the media at the new location using the If-Match header.

4.9. Media Properties

When an RTSP server handles media, it is important to consider the different properties a media instance for delivery and playback can have. This specification considers the below listed media properties in its protocol operations. They are derived from the differences between a number of supported usages.

On-demand: Media that has a fixed (given) duration that doesn't change during the life time of the RTSP session and is known at the time of the creation of the session. It is expected that the content of the media will not change, even if the representation, i.e encoding, quality, etc, may change. Generally one can seek, i.e. request any range, within the media.

Dynamic On-demand: This is a variation of the on-demand case where external methods are used to manipulate the actual content of the media setup for the RTSP session. The main example is a content defined by a playlist.

Live: Live media represents a progressing content stream (such as broadcast TV) where the duration may or may not be known. It is not seekable, only the content presently being delivered can be accessed.

Live with Recording: A Live stream that is combined with a server-side capability to store and retain the content of the live session, and allow for random access delivery within the part of the already recorded content. The actual behavior of the media stream is very much dependent on the retention policy for the media stream; either the server will be able to capture the complete media stream, or it will have a limitation in how much will be retained. The media range will dynamically change as the session progress. For servers with a limited amount of storage available for recording, there will typically be a sliding window

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1325
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

that moves forwards while new data is made available and older data is discarded.

To cover the above usages, the following media properties with appropriate values are specified:

4.9.1. Random Access and Seeking

Random Access is the ability to specify and get media delivered from any point inside the content, an operation called seeking. This possibility is signaled using the Seek-Style header (see Section Section 16.45) which can take the following different values:

Random Access: The media are seekable to any out of a large number of points within the media. Due to media encoding limitations, a particular point may not be reachable, but seeking to a point close by is enabled. A floating point number of seconds may be provided to express the worst case distance between random access points.

Conditional Random Access: Based on the above Random Access but intended to handle a case where the distance in the media between random access points are large, and where small seek forward using Random Access would move the client further away then the current point.

Return To Start: Seeking is only possible to the beginning of the content.

No seeking: Seeking is not possible at all.

4.9.2. Retention

Media may have different retention policies in place that affect the operation on media. The following different media retention policies are envisioned and taken into consideration where applicable:

Unlimited: The media will not be removed as long as the RTSP session is in existence.

Time Limited: The media will not be removed before given wallclock time. After that time it may or may not be available any more.

Duration limited: Each individual unit of the media will be retained for the specified duration.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1326
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

4.9.3. Content Modifications

There is also the question of how the content may change during time for a give media resource:

Immutable: The content of the media will not change, even if the representation, i.e., encoding, quality, etc., may change.

Dynamic: Between explicit updates the media content will not change, but the content may change due to external methods or triggers, such as playlists.

Time Progressing: As times progresses new content will become available. If the content also is retained it will become longer as everything between the start point and the point currently being made available can be accessed. If the media server uses a sliding window policy for retention, the start point will also change as time progresses.

4.9.4. Supported Scale Factors

Content often supports only a limited set or range of scales when delivering the media.. To enable the client to know what values or ranges of scale operations that the whole content or the current position supports, a media properties attribute for this is defined which contains a list with the values and/or ranges that are supported. The attribute is named "Scales". It may be updated at any point in the content due to content consisting of spliced pieces or content being dynamically updated by out-of-band mechanisms.

4.9.5. Mapping to the Attributes

This section shows examples of how one would map the above usages to the properties and their values.

On-demand: Random Access: Random Access=5s, Content Modifications: Immutable, Retention: unlimited or time limited.

Dynamic On-demand: Random Access: Random Access=3s, Content Modifications: Dynamic, Retention: unlimited or time limited.

Live: Random Access: No seeking, Content Modifications: Time Progressing, Retention: Duration limited=0.0s

Live with Recording: Random Access: Random Access=3s, Content Modifications: Time Progressing, Retention: Duration limited=2H

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1327
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5. RTSP Message

RTSP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding RFC 3629 [RFC3629]. Lines MUST be terminated by CRLF.

> Text-based protocols make it easier to add optional parameters in a self-describing manner. Since the number of parameters and the frequency of commands is low, processing efficiency is not a concern. Text-based protocols, if done carefully, also allow easy implementation of research prototypes in scripting languages such as TCL, Visual Basic and Perl.

The ISO 10646 character set avoids tricky character set switching, but is invisible to the application as long as US-ASCII is being used. This is also the encoding used for RTCP [RFC3550].

Requests contain methods, the object the method is operating upon and parameters to further describe the method. Methods are idempotent unless otherwise noted. Methods are also designed to require little or no state maintenance at the media server.

5.1. Message Types

RTSP messages consist of requests from client to server, or server to client, and responses in the reverse direction. Request Section 7 and Response Section 8 messages use a format based on the generic message format of RFC 2822 [RFC2822] for transferring bodies (the payload of the message). Both types of message consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header, and possibly the data of the message-body.

```
generic-message = start-line
                 *(message-header CRLF)
                   CRLF
                  [ message-body-data ]
start-line = Request-Line | Status-Line
```

In the interest of robustness, servers MUST ignore any empty line(s) received where a Request-Line is expected. In other words, if the server is reading the protocol stream at the beginning of a message and receives a CRLF first, it should ignore the CRLF.

5.2. Message Headers

RTSP header fields (see Section 16) include general-header, request-header, response-header, and Message-body header fields.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1328
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The order in which header fields with differing field names are received is not significant. However, it is "good practice" to send general-header fields first, followed by request-header or response-header fields, and ending with the Message-body header fields.

Multiple message-header fields with the same field-name MAY be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list. It MUST be possible to combine the multiple header fields into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The order in which header fields with the same field-name are received is therefore significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

Unknown message headers MUST be ignored (skipping over the header to the next protocol element, and not causing an error) by a RTSP server or client. An RTSP Proxy MUST forward unknown message headers. Message headers defined outside of this specification that are required to be interpreted by the RTSP agent will need to use feature tags (Section 4.7) and include them in the appropriate Require (Section 16.41) or Proxy-Require (Section 16.35) header.

5.3. Message Body

The message-body (if any) of an RTSP message is used to carry further information for a particular resource associated with the request or response. An example of a message body is the Session Description Protocol (SDP).

The presence of a message-body in either a request or a response MUST be signaled by the inclusion of a Content-Length header (see Section 16.16).

The presence of a message-body in a request is signaled by the inclusion of a Content-Length header field in the RTSP message. A message-body MUST NOT be included in a request or response if the specification of the particular method (see Method Definitions (Section 13)) does not allow sending a message body.

5.4. Message Length

When a message body is included with a message, the length of that body is determined by one of the following (in order of precedence):

1. Any response message which MUST NOT include a message body (such as the 1xx, 204, and 304 responses) is always terminated by the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1329
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

first empty line after the header fields, regardless of the message-header fields present in the message. (Note: An empty line is a line with nothing preceding the CRLF.)

2. If a Content-Length header(Section 16.16) is present, its value in bytes represents the length of the message-body. If this header field is not present, a value of zero is assumed.

Unlike an HTTP message, an RTSP message MUST contain a Content-Length header whenever it contains a message body. Note that RTSP does not support the HTTP/1.1 "chunked" transfer coding (see [H3.6.1]).

Given the moderate length of presentation descriptions returned, the server should always be able to determine its length, even if it is generated dynamically, making the chunked transfer encoding unnecessary.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1330
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

6. General Header Fields

General headers are headers that may be used in both requests and responses. The general headers are listed in Table 1:

Header Name	Defined in Section
Accept-Ranges	Section 16.5
Cache-Control	Section 16.10
Connection	Section 16.11
CSeq	Section 16.19
Date	Section 16.20
Media-Properties	Section 16.28
Media-Range	Section 16.29
Pipelined-Requests	Section 16.32
Proxy-Supported	Section 16.36
RTP-Info	Section 16.43
Seek-Style	Section 16.45
Supported	Section 16.49
Timestamp	Section 16.51
Via	Section 16.56

Table 1: The general headers used in RTSP

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1331
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

7. Request

A request message uses the format outlined below regardless of the direction of a request, client to server or server to client:

- Request line, containing the method to be applied to the resource, the identifier of the resource, and the protocol version in use;
- Zero or more Header lines, that can be of the following types: general (Section 6), request (Section 7.2), or message body(Section 9.1);
- One empty line (CRLF) to indicate the end of the header section;
- Optionally a message-body, consisting of one or more lines. The length of the message body in bytes is indicated by the Content-Length message header.

7.1. Request Line

The request line provides the key information about the request: what method, on what resources and using which RTSP version. The methods that are defined by this specification are listed in Table 2.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1332
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Method	Defined in Section
DESCRIBE	Section 13.2
GET_PARAMETER	Section 13.8
OPTIONS	Section 13.1
PAUSE	Section 13.6
PLAY	Section 13.4
PLAY_NOTIFY	Section 13.5
REDIRECT	Section 13.10
SETUP	Section 13.3
SET_PARAMETER	Section 13.9
TEARDOWN	Section 13.7

Table 2: The RTSP Methods

The syntax of the RTSP request line is the following:

<Method> <Request-URI> <RTSP-Version> CRLF

Note: This syntax cannot be freely changed in future versions of RTSP. This line needs to remain parsable by older RTSP implementations since it indicates the RTSP version of the message.

In contrast to HTTP/1.1 [RFC2616], RTSP requests identify the resource through an absolute RTSP URI (including scheme, host, and port) (see Section 4.2) rather than just the absolute path.

HTTP/1.1 requires servers to understand the absolute URI, but clients are supposed to use the Host request header. This is purely needed for backward-compatibility with HTTP/1.0 servers, a consideration that does not apply to RTSP.

An asterisk "*" can be used instead of an absolute URI in the Request-URI part to indicate that the request does not apply to a particular resource, but to the server or proxy itself, and is only allowed when the request method does not necessarily apply to a resource.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1333
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

For example:

```
OPTIONS * RTSP/2.0
```

An OPTIONS in this form will determine the capabilities of the server or the proxy that first receives the request. If the capability of the specific server needs to be determined, without regard to the capability of an intervening proxy, the server should be addressed explicitly with an absolute URI that contains the server's address.

For example:

```
OPTIONS rtsp://example.com RTSP/2.0
```

7.2. Request Header Fields

The RTSP headers in Table 3 can be included in a request, as request headers, to modify the specifics of the request. Some of these headers may also be used in the response to a request, as response headers, to modify the specifics of a response (Section 8.2).

Header	Defined in Section
Accept	Section 16.1
Accept-Credentials	Section 16.2
Accept-Encoding	Section 16.3
Accept-Language	Section 16.4
Authorization	Section 16.7
Bandwidth	Section 16.8
Blocksize	Section 16.9
From	Section 16.22
If-Match	Section 16.23
If-Modified-Since	Section 16.24
If-None-Match	Section 16.25
Notify-Reason	Section 16.31

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1334
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Proxy-Require	Section 16.35
Range	Section 16.38
Terminate-Reason	Section 16.50
Referrer	Section 16.39
Request-Status	Section 16.40
Require	Section 16.41
Scale	Section 16.44
Session	Section 16.47
Speed	Section 16.48
Supported	Section 16.49
Transport	Section 16.52
User-Agent	Section 16.54

Table 3: The RTSP request headers

Detailed header definition are provided in Section 16.

New request headers may be defined. If the receiver of the request is required to understand the request header, the request MUST include a corresponding feature tag in a Require or Proxy-Require header to ensure the processing of the header.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1335
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

8. Response

After receiving and interpreting a request message, the recipient responds with an RTSP response message. Normally, there is only one, final, response. Only responses using the response code class 1xx, that it is allowed to send one or more 1xx response messages prior to the final response message.

The valid response codes and the methods they can be used with are listed in Table 4.

8.1. Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and the textual phrase associated with the status code, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

<RTSP-Version> SP <Status-Code> SP <Reason-Phrase> CRLF

8.1.1. Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in Section 15. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

1xx: Informational - Request received, continuing process

2xx: Success - The action was successfully received, understood, and accepted

3rr: Redirection - Further action needs to be taken in order to complete the request

4xx: Client Error - The request contains bad syntax or cannot be fulfilled

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1336
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

5xx: Server Error - The server failed to fulfill an apparently valid request

The individual values of the numeric status codes defined for RTSP/2.0, and an example set of corresponding Reason-Phrases, are presented in Table 4. The reason phrases listed here are only recommended; they may be replaced by local equivalents without affecting the protocol. Note that RTSP adopts most HTTP/1.1 [RFC2616] status codes and adds RTSP-specific status codes starting at x50 to avoid conflicts with future HTTP status codes that are desirable to import into RTSP.

RTSP status codes are extensible. RTSP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications MUST understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response MUST NOT be cached. For example, if an unrecognized status code of 431 is received by the client, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents SHOULD present to the user the message body returned with the response, since that message body is likely to include human-readable information which will explain the unusual status.

Code	Reason	Method
100	Continue	all
200	OK	all
301	Moved Permanently	all
302	Found	all
304	Not Modified	all
305	Use Proxy	all
400	Bad Request	all
401	Unauthorized	all

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1337
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

402	Payment Required	all
403	Forbidden	all
404	Not Found	all
405	Method Not Allowed	all
406	Not Acceptable	all
407	Proxy Authentication Required	all
408	Request Timeout	all
410	Gone	all
411	Length Required	all
412	Precondition Failed	DESCRIBE, SETUP
413	Request Message Body Too Large	all
414	Request-URI Too Long	all
415	Unsupported Media Type	all
451	Parameter Not Understood	SET_PARAMETER, GET_PARAMETER
452	reserved	n/a
453	Not Enough Bandwidth	SETUP
454	Session Not Found	all
455	Method Not Valid In This State	all
456	Header Field Not Valid	all
457	Invalid Range	PLAY, PAUSE
458	Parameter Is Read-Only	SET_PARAMETER
459	Aggregate Operation Not Allowed	all
460	Only Aggregate Operation Allowed	all

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1338
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

461	Unsupported Transport	all
462	Destination Unreachable	all
463	Destination Prohibited	SETUP
464	Data Transport Not Ready Yet	PLAY
465	Notification Reason Unknown	PLAY_NOTIFY
470	Connection Authorization Required	all
471	Connection Credentials not accepted	all
472	Failure to establish secure connection	all
500	Internal Server Error	all
501	Not Implemented	all
502	Bad Gateway	all
503	Service Unavailable	all
504	Gateway Timeout	all
505	RTSP Version Not Supported	all
551	Option Not Support	all

Table 4: Status codes and their usage with RTSP methods

8.2. Response Headers

The response-header allows the request recipient to pass additional information about the response which cannot be placed in the Status-Line. This header give information about the server and about further access to the resource identified by the Request-URI. All headers currently classified as response headers are listed in Table 5.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1339
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Header	Defined in Section
Connection-Credentials	Section 16.12
MTag	Section 16.30
Location	Section 16.27
Proxy-Authenticate	Section 16.33
Public	Section 16.37
Range	Section 16.38
Retry-After	Section 16.42
Scale	Section 16.44
Session	Section 16.47
Server	Section 16.46
Speed	Section 16.48
Transport	Section 16.52
Unsupported	Section 16.53
Vary	Section 16.55
WWW-Authenticate	Section 16.57

Table 5: The RTSP response headers

Response-header names can be extended reliably only in combination with a change in the protocol version. However, the usage of feature-tags in the request allows the responding party to learn the capability of the receiver of the response. A new or experimental header MAY be given the semantics of response-header if all parties in the communication recognize them to be response-header. Unrecognized headers in responses are treated as message-headers.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1340
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

9. Message Body

Request and Response messages MAY transfer a message body, if not otherwise restricted by the request method or response status code. The message body consists of message-body header fields and the content data itself.

The SET_PARAMETER and GET_PARAMETER request and response, and DESCRIBE response MAY have an message body. All 4xx and 5xx responses MAY also have an message body.

In this section, both sender and recipient refer to either the client or the server, depending on who sends and who receives the message body.

9.1. Message-Body Header Fields

Message-body header fields define meta-information about the content data in the message body. The message-body header fields are listed in Table 6.

Header	Defined in Section
Allow	Section 16.6
Content-Base	Section 16.13
Content-Encoding	Section 16.14
Content-Language	Section 16.15
Content-Length	Section 16.16
Content-Location	Section 16.17
Content-Type	Section 16.18
Expires	Section 16.21
Last-Modified	Section 16.26

Table 6: The RTSP message-body headers

The extension-header mechanism allows additional message-body header fields to be defined without changing the protocol, but these fields cannot be assumed to be recognizable by the recipient. Unrecognized

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1341
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

header fields MUST be ignored by the recipient and forwarded by proxies.

9.2. Message Body

RTSP message with an message body MUST include the Content-Type and Content-Length headers. When a message body is included with a message, the data type of that content data is determined via the header fields Content-Type and Content-Encoding.

Content-Type specifies the media type of the underlying data. Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource. There is no default encoding.

The Content-Length of a message is the length of the content, measured in bytes.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1342
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

10. Connections

RTSP requests can be transmitted using the two different connection scenarios listed below:

- persistent - a transport connection is used for several request/response transactions;
- transient - a transport connection is used for a single request/response transaction.

RFC 2326 attempted to specify an optional mechanism for transmitting RTSP messages in connectionless mode over a transport protocol such as UDP. However, it was not specified in sufficient detail to allow for interoperable implementations. In an attempt to reduce complexity and scope, and due to lack of interest, RTSP 2.0 does not attempt to define a mechanism for supporting RTSP over UDP or other connectionless transport protocols. A side-effect of this is that RTSP requests MUST NOT be sent to multicast groups since no connection can be established with a specific receiver in multicast environments.

Certain RTSP headers, such as the CSeq header (Section 16.19), which may appear to be relevant only to connectionless transport scenarios are still retained and must be implemented according to the specification. In the case of CSeq, it is quite useful for matching responses to requests if the requests are pipelined (see Section 12). It is also useful in proxies for keeping track of the different requests when aggregating several client requests on a single TCP connection.

10.1. Reliability and Acknowledgements

Since RTSP messages are transmitted using reliable transport protocols, they MUST NOT be retransmitted at the RTSP protocol level. Instead, the implementation must rely on the underlying transport to provide reliability. The RTSP implementation may use any indication of reception acknowledgment of the message from the underlying transport protocols to optimize the RTSP behavior.

> If both the underlying reliable transport such as TCP and the RTSP application retransmit requests, each packet loss or message loss may result in two retransmissions. The receiver typically cannot take advantage of the application-layer retransmission since the transport stack will not deliver the application-layer retransmission before the first attempt has reached the receiver. If the packet loss is caused by congestion, multiple retransmissions at different layers will exacerbate the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1343
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

congestion.

Lack of acknowledgment of an RTSP request should be handled within the constraints of the connection timeout considerations described below (Section 10.4).

10.2. Using Connections

A TCP transport can be used for both persistent connections (for several message exchanges) and transient connections (for a single message exchange). Implementations of this specification MUST support RTSP over TCP. The scheme of the RTSP URI (Section 4.2) indicates the default port that the server will listen on if the port is not explicitly given.

A server MUST handle both persistent and transient connections.

> Transient connections facilitate mechanisms for fault tolerance. They also allow for application layer mobility. A server and client pair that support transient connections can survive the loss of a TCP connection; e.g., due to a NAT timeout. When the client has discovered that the TCP connection has been lost, it can set up a new one when there is need to communicate again.

A persistent connection is RECOMMENDED to be used for all transactions between the server and client, including messages for multiple RTSP sessions. However, a persistent connection MAY be closed after a few message exchanges. For example, a client may use a persistent connection for the initial SETUP and PLAY message exchanges in a session and then close the connection. Later, when the client wishes to send a new request, such as a PAUSE for the session, a new connection would be opened. This connection may either be transient or persistent.

An RTSP agent SHOULD NOT have more than one connection to the server at any given point. If a client or proxy handles multiple RTSP sessions on the same server, it SHOULD use only one connection for managing those sessions.

> This saves connection resources on the server. It also reduces complexity by enabling the server to maintain less state about its sessions and connections.

RTSP allows a server to send requests to a client. However, this can be supported only if a client establishes a persistent connection with the server. In cases where a persistent connection does not exist between a server and its client, due to the lack of a signaling channel the server may be forced to silently discard RTSP messages,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1344
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

and may even drop an RTSP session without notifying the client. An example of such a case is when the server desires to send a REDIRECT request for an RTSP session to the client but is not able to do so because it cannot reach the client. A server that attempts to send a request to a client that has no connection currently to the server SHOULD discard the request directly, but it MAY queue it for later delivery. However, if the server queues the request it should when adding additional requests to the queue ensure to remove older requests that are now redundant.

> Without a persistent connection between the client and the server, the media server has no reliable way of reaching the client. Because the likely failure of server to client established connections the server will not even attempt establishing any connection.

The sending of client and server requests can be asynchronous events. To avoid deadlock situations both client and server MUST be able to send and receive requests simultaneously. As an RTSP response may be queued up for transmission, reception or processing behind the peer RTSP agent's own requests, all RTSP agents are required to have a certain capability of handling outstanding messages. A potential issue is that outstanding requests may timeout despite them being processed by the peer due to the response is caught in the queue behind a number of request that the RTSP agent is processing but that take some time to complete. To avoid this problem an RTSP agent is recommended to buffer incoming messages locally so that any response messages can be processed immediately upon reception. If responses are separated from requests and directly forwarded for processing, not only the result be used immediately, the state associated with that outstanding request can also be released. However, buffering a number of requests on the receiving RTSP agent consumes resources and enables a resource exhaustion attack on the agent. Therefore this buffer should be limited so that an unreasonable number of requests or total message size is not allowed to consume the receiving agent's resources. In most APIs having the receiving agent stop reading from the TCP socket will result in TCP's window being clamped. Thus forcing the buffering onto the sending agent when the load is larger than expected. However, as both RTSP message sizes and frequency may be changed in the future by protocol extensions, an agent should be careful against taking harsher measurements against a potential attack. When under attack an RTSP agent can close TCP connections and release state associated with that TCP connection.

To provide some guidance on what is reasonable the following guidelines are given. An RTSP agent should not have more than 10 outstanding requests per RTSP session. An RTSP agent should not have more than 10 outstanding requests that aren't related to an RTSP

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1345
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

session or that are requesting to create an RTSP session.

In light of the above, it is RECOMMENDED that clients use persistent connections whenever possible. A client that supports persistent connections MAY "pipeline" its requests (see Section 12).

10.3. Closing Connections

The client MAY close a connection at any point when no outstanding request/response transactions exist for any RTSP session being managed through the connection. The server, however, SHOULD NOT close a connection until all RTSP sessions being managed through the connection have been timed out (Section 16.47). A server SHOULD NOT close a connection immediately after responding to a session-level TEARDOWN request for the last RTSP session being controlled through the connection. Instead, it should wait for a reasonable amount of time for the client to receive the TEARDOWN response, take appropriate action, and initiate the connection closing. The server SHOULD wait at least 10 seconds after sending the TEARDOWN response before closing the connection.

> This is to ensure that the client has time to issue a SETUP for a new session on the existing connection after having torn the last one down. 10 seconds should give the client ample opportunity to get its message to the server.

A server SHOULD NOT close the connection directly as a result of responding to a request with an error code.

> Certain error responses such as "460 Only Aggregate Operation Allowed" (Section 15.4.25) are used for negotiating capabilities of a server with respect to content or other factors. In such cases, it is inefficient for the server to close a connection on an error response. Also, such behavior would prevent implementation of advanced/special types of requests or result in extra overhead for the client when testing for new features. On the flip side, keeping connections open after sending an error response poses a Denial of Service security risk (Section 21).

The server MAY close a connection if he receives an incomplete message and if the message is not completed within a reasonable amount of time. It is RECOMMENDED that the server waits at least 10 second for the completion of a message or for the next part of the message to arrive (which is an indication that the transport and the client are still alive). Servers believing they are under attack or otherwise starved for resources during that event consider using a shorter timeout.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1346
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

If a server closes a connection while the client is attempting to send a new request, the client will have to close its current connection, establish a new connection and send its request over the new connection.

An RTSP message should not be terminated by closing the connection. Such a message MAY be considered to be incomplete by the receiver and discarded. An RTSP message is properly terminated as defined in Section 5.

10.4. Timing Out Connections and RTSP Messages

Receivers of a request (responder) SHOULD respond to requests in a timely manner even when a reliable transport such as TCP is used. Similarly, the sender of a request (requester) SHOULD wait for a sufficient time for a response before concluding that the responder will not be acting upon its request.

A responder SHOULD respond to all requests within 5 seconds. If the responder recognizes that processing of a request will take longer than 5 seconds, it SHOULD send a 100 (Continue) response as soon as possible. It SHOULD continue sending a 100 response every 5 seconds thereafter until it is ready to send the final response to the requester. After sending a 100 response, the receiver MUST send a final response indicating the success or failure of the request.

A requester SHOULD wait at least 10 seconds for a response before concluding that the responder will not be responding to its request. After receiving a 100 response, the requester SHOULD continue waiting for further responses. If more than 10 seconds elapses without receiving any response, the requester MAY assume that the responder is unresponsive and abort the connection.

A requester SHOULD wait longer than 10 seconds for a response if it is experiencing significant transport delays on its connection to the responder. The requester is capable of determining the RTT of the request/response cycle using the Timestamp header (Section 16.51) in any RTSP request.

> 10 seconds was chosen for the following reasons. It gives TCP time to perform a couple of retransmissions, even if operating on default values. It is short enough that users may not abandon the process themselves. However, it should be noted that 10 seconds can be aggressive on certain type of networks. The 5 seconds value for 1xx messages is half the timeout giving a reasonable change of successful delivery before timeout happens on the requester side.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1347
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

10.5. Showing Liveness

The mechanisms for showing liveness of the client is, any RTSP request with a Session header, if RTP & RTCP is used an RTCP message, or through any other used media protocol capable of indicating liveness of the RTSP client. It is RECOMMENDED that a client does not wait to the last second of the timeout before trying to send a liveness message. The RTSP message may be lost or when using reliable protocols, such as TCP, the message may take some time to arrive safely at the receiver. To show liveness between RTSP request issued to accomplish other things, the following mechanisms can be used, in descending order of preference:

RTCP: If RTP is used for media transport RTCP SHOULD be used. If RTCP is used to report transport statistics, it MUST also work as keep alive. The server can determine the client by network address and port together with the fact that the client is reporting on the servers SSRC(s). A downside of using RTCP is that it only gives statistical guarantees to reach the server. However, the probability of a false client timeout is so low that it can be ignored in most cases. For example, assume a session with 60 seconds timeout and enough bitrate assigned to RTCP messages to send a message from client to server on average every 5 seconds. That client have, for a network with 5 % packet loss, the probability to fail showing liveness sign in that session within the timeout interval of 2.4*E-16. In sessions with shorter timeouts, or much higher packet loss, or small RTCP bandwidths SHOULD also use any of the mechanisms below.

SET_PARAMETER: When using SET_PARAMETER for keep alive, no body SHOULD be included. This method is the RECOMMENDED RTSP method to use for a request intended only to perform keep-alive.

GET_PARAMETER: When using GET_PARAMETER for keep alive, no body SHOULD be included.

OPTIONS: This method is also usable, but it causes the server to perform more unnecessary processing and result in bigger responses than necessary for the task. The reason is that the server needs to determine the capabilities associated with the media resource to correctly populate the Public and Allow headers.

The timeout parameter MAY be included in a SETUP response, and MUST NOT be included in requests. The server uses it to indicate to the client how long the server is prepared to wait between RTSP commands or other signs of life before closing the session due to lack of

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1348
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

activity (see Appendix B). The timeout is measured in seconds, with a default of 60 seconds. The length of the session timeout MUST NOT be changed in an established session.

10.6. Use of IPv6

Explicit IPv6 support was not present in RTSP 1.0 (RFC 2326). RTSP 2.0 has been updated for explicit IPv6 support. Implementations of RTSP 2.0 MUST understand literal IPv6 addresses in URIs and headers.

10.7. Overload Control

Overload in RTSP can occur when server and proxies have insufficient resources to complete the processing of a request. An improper handling of such an overload situation at proxies and servers can impact the operation of the RTSP deployment, probably worsen the situation. RTSP defines the 503 (Service Unavailable) response (Section Section 15.5.4) to let servers and proxies notify requesting proxies and RTSP clients about an overload situation. In conjunction with the Retry-After header (Section Section 16.42) the server or proxy can indicate the time after the requesting entity can send another request to the proxy or server.

Simply implementing and using the 503 (Service Unavailable) is not sufficient enough for properly handling overload situations. For instance, a simplistic approach would be to send the 503 response with a Retry-After header set to a fixed value. However, this can cause the situation where multiple RTSP clients again send requests to a proxy or server at roughly the same time which may again cause an overload situation, or if the "old" overload situation is not yet solved, i.e., the length indicated in the Retry-After header was too short.

An RTSP server or proxy in an overload situation must select the value of the Retry-After header carefully and in dependency of its current load situation. It is RECOMMENDED to increase the length proportional with the current load of the server, i.e., an increasing workload should result in an increased length of the indicated unavailability. It is RECOMMENDED to not send the same value in the Retry-After header to all requesting proxies and clients, but to add a variation the mean value of the Retry-After header.

Another issue are load balancing RTSP proxies, i.e., where an RTSP proxy is used to select amongst a set of RTSP servers to handled the requests, or when multiple server addresses are available for a given server name. The proxy or client may receive a 503 (Service Unavailable) from one of its RTSP servers or a TCP timeout (if the server is even unable to handled the request message). The proxy or

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1349
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

client simply retries the other addresses, but may also receive a 503 (Service Unavailable) response or TCP timeouts from those addresses. In such a situation, where none of the RTSP servers/addresses can handled the request, the RTSP agent has to wait before it can send any new requests to the RTSP server. Any additional request to a specific address MUST be delayed according to the Retry-After headers received. For addresses where no response was received or TCP timeout occurred, an initial wait timer SHOULD be set to 5 seconds. That timer MUST be doubled for each additional failure to connect or receive response.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1350
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

11. Capability Handling

This section describes the available capability handling mechanism which allows RTSP to be extended. Extensions to this version of the protocol are basically done in two ways. First, new headers can be added. Secondly, new methods can be added. The capability handling mechanism is designed to handle both cases.

When a method is added, the involved parties can use the OPTIONS method to discover whether it is supported. This is done by issuing a OPTIONS request to the other party. Depending on the URI it will either apply in regards to a certain media resource, the whole server in general, or simply the next hop. The OPTIONS response MUST contain a Public header which declares all methods supported for the indicated resource.

It is not necessary to use OPTIONS to discover support of a method, as the client could simply try the method. If the receiver of the request does not support the method it will respond with an error code indicating the method is either not implemented (501) or does not apply for the resource (405). The choice between the two discovery methods depends on the requirements of the service.

Feature-Tags are defined to handle functionality additions that are not new methods. Each feature-tag represents a certain block of functionality. The amount of functionality that a feature-tag represents can vary significantly. A feature-tag can for example represent the functionality a single RTSP header provides. Another feature-tag can represent much more functionality, such as the "play.basic" feature-tag which represents the minimal media delivery for playback implementation.

Feature-tags are used to determine whether the client, server or proxy supports the functionality that is necessary to achieve the desired service. To determine support of a feature-tag, several different headers can be used, each explained below:

Supported: This header is used to determine the complete set of functionality that both client and server have. The intended usage is to determine before one needs to use a functionality that it is supported. It can be used in any method, but OPTIONS is the most suitable one as it at the same time determines all methods that are implemented. When sending a request the requester declares all its capabilities by including all supported feature-tags. This results in the receiver learns the requester's feature support. The receiver then includes its set of features in the response.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1351
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Proxy-Supported: This header is used similarly to the Supported header, but instead of giving the supported functionality of the client or server it provides both the requester and the responder a view of what functionality the proxy chain between the two supports. Proxies are required to add this header whenever the Supported header is present, but proxies may also add it independently of the requester.

Require: This header can be included in any request where the end-point, i.e. the client or server, is required to understand the feature to correctly perform the request. This can, for example, be a SETUP request where the server is required to understand a certain parameter to be able to set up the media delivery correctly. Ignoring this parameter would not have the desired effect and is not acceptable. Therefore the end-point receiving a request containing a Require MUST negatively acknowledge any feature that it does not understand and not perform the request. The response in cases where features are not supported are 551 (Option Not Supported). Also the features that are not supported are given in the Unsupported header in the response.

Proxy-Require: This header has the same purpose and workings as Require except that it only applies to proxies and not the end-point. Features that need to be supported by both proxies and end-points need to be included in both the Require and Proxy-Require header.

Unsupported: This header is used in a 551 error response, to indicate which features were not supported. Such a response is only the result of the usage of the Require and/or Proxy-Require header where one or more feature where not supported. This information allows the requester to make the best of situations as it knows which features are not supported.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1352
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

12. Pipelining Support

Pipelining is a general method to improve performance of request response protocols by allowing the requesting agent to have more than one request outstanding and send them over the same persistent connection. For RTSP, where the relative order of requests will matter, it is important to maintain the order of the requests. Because of this, the responding agent MUST process the incoming requests in their sending order. The sending order can be determined by the CSeq header and its sequence number. For TCP the delivery order will be the same as the sending order. The processing of the request MUST also have been finished before processing the next request from the same agent. The responses MUST be sent in the order the requests were processed.

RTSP 2.0 has extended support for pipelining compared to RTSP 1.0. The major improvement is to allow all requests to setup and initiate media delivery to be pipelined after each other. This is accomplished by the utilization of the Pipelined-Requests header (see Section 16.32). This header allows a client to request that two or more requests are processed in the same RTSP session context which the first request creates. In other words, a client can request that two or more media streams are set-up and then played without needing to wait for a single response. This speeds up the initial startup time for an RTSP session with at least one RTT.

If a pipelined request builds on the successful completion of one or more prior requests the requester must verify that all requests were executed as expected. A common example will be two SETUP requests and a PLAY request. In case one of the SETUP fails unexpectedly, the PLAY request can still be successfully executed. However, the resulting presentation will not be as expected by the requesting client, as only a single media instead of two will be played. In this case the client can send a PAUSE request, correct the failing SETUP request and then request it to be played.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1353
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

13. Method Definitions

The method indicates what is to be performed on the resource identified by the Request-URI. The method name is case-sensitive. New methods may be defined in the future. Method names MUST NOT start with a $ character (decimal 36) and MUST be a token as defined by the ABNF [RFC5234] in the syntax chapter Section 20. The methods are summarized in Table 7.

method	direction	object	Server req.	Client req.
DESCRIBE	C -> S	P,S	recommended	recommended
GET_PARAMETER	C -> S	P,S	optional	optional
	S -> C	P,S	optional	optional
OPTIONS	C -> S	P,S	required	required
	S -> C	P,S	optional	optional
PAUSE	C -> S	P,S	required	required
PLAY	C -> S	P,S	required	required
PLAY_NOTIFY	S -> C	P,S	required	required
REDIRECT	S -> C	P,S	optional	required
SETUP	C -> S	S	required	required
SET_PARAMETER	C -> S	P,S	required	optional
	S -> C	P,S	optional	optional
TEARDOWN	C -> S	P,S	required	required
	S -> C	P	required	required

Table 7: Overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1354
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Note on Table 7: GET_PARAMETER is recommended, but not required. For example, a fully functional server can be built to deliver media without any parameters. SET_PARAMETER is required, however, due to its usage for keep-alive. PAUSE is now required because it is the only way of leaving the Play state without terminating the whole session.

If an RTSP agent does not support a particular method, it MUST return 501 (Not Implemented) and the requesting RTSP agent, in turn, SHOULD NOT try this method again for the given agent / resource combination. An RTSP proxy whose main function is to log or audit and not modify transport or media handling in any way MAY forward RTSP messages with unknown methods. Note that the proxy still needs to perform the minimal required processing, like adding the Via header.

13.1. OPTIONS

The semantics of the RTSP OPTIONS method is similar to that of the HTTP OPTIONS method described in [H9.2]. In RTSP however, OPTIONS is bi-directional, in that a client can request it to a server and vice versa. A client MUST implement the capability to send an OPTIONS request and a server or a proxy MUST implement the capability to respond to an OPTIONS request. The client, server or proxy MAY also implement the converse of their required capability.

An OPTIONS request may be issued at any time. Such a request does not modify the session state. However, it may prolong the session lifespan (see below). The URI in an OPTIONS request determines the scope of the request and the corresponding response. If the Request-URI refers to a specific media resource on a given host, the scope is limited to the set of methods supported for that media resource by the indicated RTSP agent. A Request-URI with only the host address limits the scope to the specified RTSP agent's general capabilities without regard to any specific media. If the Request-URI is an asterisk ("*"), the scope is limited to the general capabilities of the next hop (i.e. the RTSP agent in direct communication with the request sender).

Regardless of scope of the request, the Public header MUST always be included in the OPTIONS response listing the methods that are supported by the responding RTSP agent. In addition, if the scope of the request is limited to a media resource, the Allow header MUST be included in the response to enumerate the set of methods that are allowed for that resource unless the set of methods completely matches the set in the Public header. If the given resource is not available, the RTSP agent SHOULD return an appropriate response code such as 3rr or 4xx. The Supported header MAY be included in the request to query the set of features that are supported by the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1355
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

responding RTSP agent.

The OPTIONS method can be used to keep an RTSP session alive. However, this is not the preferred way of session keep-alive signaling, see Section 16.47. An OPTIONS request intended for keeping alive an RTSP session MUST include the Session header with the associated session ID. Such a request SHOULD also use the media or the aggregated control URI as the Request-URI.

Example:

```
  C->S:  OPTIONS rtsp://server.example.com RTSP/2.0
         CSeq: 1
         User-Agent: PhonyClient/1.2
         Proxy-Require: gzipped-messages
         Supported: play.basic

  S->C:  RTSP/2.0 200 OK
         CSeq: 1
         Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE, OPTIONS
         Supported: play.basic, setup.rtp.rtcp.mux, play.scale
         Server: PhonyServer/1.1
```

Note that some of the feature-tags in Supported and Proxy-Require are fictional features.

13.2. DESCRIBE

The DESCRIBE method is used to retrieve the description of a presentation or media object from a server. The Request-URI of the DESCRIBE request identifies the media resource of interest. The client MAY include the Accept header in the request to list the description formats that it understands. The server MUST respond with a description of the requested resource and return the description in the message body of the response, if the DESCRIBE method request can be successfully fulfilled. The DESCRIBE reply-response pair constitutes the media initialization phase of RTSP.

The DESCRIBE response SHOULD contain all media initialization information for the resource(s) that it describes. Servers SHOULD NOT use the DESCRIBE response as a means of media indirection by having the description point at another server; instead, using the 3rr responses is RECOMMENDED.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1356
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

By forcing a DESCRIBE response to contain all media initialization for the set of streams that it describes, and discouraging the use of DESCRIBE for media indirection, any looping problems can be avoided that might have resulted from other approaches.

Example:

```
C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/2.0
      CSeq: 312
      User-Agent: PhonyClient/1.2
      Accept: application/sdp, application/example

S->C: RTSP/2.0 200 OK
      CSeq: 312
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Server: PhonyServer/1.1
      Content-Base: rtsp://server.example.com/fizzle/foo/
      Content-Type: application/sdp
      Content-Length: 358

      v=0
      o=mhandley 2890844526 2890842807 IN IP4 192.0.2.46
      s=SDP Seminar
      i=A Seminar on the session description protocol
      u=http://www.example.com/lectures/sdp.ps
      e=seminar@example.com (Seminar Management)
      c=IN IP4 0.0.0.0
      a=control:*
      t=2873397496 2873404696
      m=audio 3456 RTP/AVP 0
      a=control:audio
      m=video 2232 RTP/AVP 31
      a=control:video
```

Media initialization is a requirement for any RTSP-based system, but the RTSP specification does not dictate that this is required to be done via the DESCRIBE method. There are three ways that an RTSP client may receive initialization information:

- via an RTSP DESCRIBE request
- via some other protocol (HTTP, email attachment, etc.)
- via some form of user interface

If a client obtains a valid description from an alternate source, the client MAY use this description for initialization purposes without issuing a DESCRIBE request for the same media. The client should use

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1357
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

any MTag to either validate the presentation description or make the session establishment conditional on being valid.

It is RECOMMENDED that minimal servers support the DESCRIBE method, and highly recommended that minimal clients support the ability to act as "helper applications" that accept a media initialization file from a user interface, and/or other means that are appropriate to the operating environment of the clients.

13.3. SETUP

The SETUP request for an URI specifies the transport mechanism to be used for the streamed media. The SETUP method may be used in two different cases; Create an RTSP session and change the transport parameters of already set up media stream. SETUP can be used in all three states; Init, and Ready, for both purposes and in PLAY to change the transport parameters. There is also a third possible usage for the SETUP method which is not specified in this memo: adding a media to a session. Using SETUP to add media to an existing session, when the session is in Play state, is unspecified.

The Transport header, see Section 16.52, specifies the media transport parameters acceptable to the client for data transmission; the response will contain the transport parameters selected by the server. This allows the client to enumerate in descending order of preference the transport mechanisms and parameters acceptable to it, while the server can select the most appropriate. It is expected that the session description format used will enable the client to select a limited number possible configurations that are offered to the server to choose from. All transport related parameters shall be included in the Transport header; the use of other headers for this purpose is discouraged due to middleboxes, such as firewalls or NATs.

For the benefit of any intervening firewalls, a client MUST indicate the known transport parameters, even if it has no influence over these parameters, for example, where the server advertises a fixed multicast address as destination.

> Since SETUP includes all transport initialization information, firewalls and other intermediate network devices (which need this information) are spared the more arduous task of parsing the DESCRIBE response, which has been reserved for media initialization.

The client MUST include the Accept-Ranges header in the request indicating all supported unit formats in the Range header. This allows the server to know which format it may use in future session related responses, such as a PLAY response without any range in the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1358
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

request. If the client does not support a time format necessary for the presentation the server MUST respond using 456 (Header Field Not Valid for Resource) and include the Accept-Ranges header with the range unit formats supported for the resource.

In a SETUP response the server MUST include the Accept-Ranges header (see Section 16.5) to indicate which time formats are acceptable to use for this media resource.

The SETUP response 200 OK MUST include the Media-Properties header (see Section 16.28). The combination of the parameters of the Media-Properties header indicate the nature of the content present in the session (see also Section 4.9). For example, a live stream with time shifting is indicated by

- Random Access set to Random-Access,
- Content Modifications set to Time Progressing,
- Retention set to Time-Duration (with specific recording window time value).

The SETUP response 200 OK MUST include the Media-Range header (see Section 16.29) if the media is Time-Progressing.

A basic example for SETUP:

```
  C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/2.0
        CSeq: 302
        Transport: RTP/AVP;unicast;dest_addr=":4588"/":4589",
                   RTP/AVP/TCP;unicast;interleaved=0-1
        Accept-Ranges: NPT, UTC
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 302
        Date: Thu, 23 Jan 1997 15:35:06 GMT
        Server: PhonyServer/1.1
        Session: 47112344;timeout=60
        Transport: RTP/AVP;unicast;dest_addr="192.0.2.53:4588"/
                   "192.0.2.53:4589"; src_addr="198.51.100.241:6256"/
                   "198.51.100.241:6257"; ssrc=2A3F93ED
        Accept-Ranges: NPT
        Media-Properties: Random-Access=3.2, Time-Progressing,
                          Time-Duration=3600.0
        Media-Range: npt=0-2893.23
```

In the above example the client wants to create an RTSP session

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1359
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

containing the media resource "rtsp://example.com/foo/bar/baz.rm". The transport parameters acceptable to the client is either RTP/AVP/UDP (UDP per default) to be received on client port 4588 and 4589 at the address the RTSP setup connection comes from or RTP/AVP interleaved on the RTSP control channel. The server selects the RTP/AVP/UDP transport and adds the address and ports it will send and received RTP and RTCP from, and the RTP SSRC that will be used by the server.

The server MUST generate a session identifier in response to a successful SETUP request, unless a SETUP request to a server includes a session identifier or an Pipelined-Requests header referencing an existing session context, in which case the server MUST bundle this setup request into the existing session (aggregated session) or return error 459 (Aggregate Operation Not Allowed) (see Section 15.4.24). An Aggregate control URI MUST be used to control an aggregated session. This URI MUST be different from the stream control URIs of the individual media streams included in the aggregate. The Aggregate control URI is to be specified by the session description if the server supports aggregated control and aggregated control is desired for the session. However, even if aggregated control is offered the client MAY chose to not set up the session in aggregated control. If an Aggregate control URI is not specified in the session description, it is normally an indication that non-aggregated control should be used. The SETUP of media streams in an aggregate which has not been given an aggregated control URI is unspecified.

> While the session ID sometimes carries enough information for aggregate control of a session, the Aggregate control URI is still important for some methods such as SET_PARAMETER where the control URI enables the resource in question to be easily identified. The Aggregate control URI is also useful for proxies, enabling them to route the request to the appropriate server, and for logging, where it is useful to note the actual resource that a request was operating on.

A session will exist until it is either removed by a TEARDOWN request or is timed-out by the server. The server MAY remove a session that has not demonstrated liveness signs from the client(s) within a certain timeout period. The default timeout value is 60 seconds; the server MAY set this to a different value and indicate so in the timeout field of the Session header in the SETUP response. For further discussion see Section 16.47. Signs of liveness for an RTSP session are:

- Any RTSP request from a client which includes a Session header with that session's ID.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1360
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- If RTP is used as a transport for the underlying media streams, an RTCP sender or receiver report from the client(s) for any of the media streams in that RTSP session. RTCP Sender Reports may for example be received in sessions where the server is invited into a conference session and is valid for keep-alive.

If a SETUP request on a session fails for any reason, the session state, as well as transport and other parameters for associated streams MUST remain unchanged from their values as if the SETUP request had never been received by the server.

13.3.1. Changing Transport Parameters

A client MAY issue a SETUP request for a stream that is already set up or playing in the session to change transport parameters, which a server MAY allow. If it does not allow changing of parameters, it MUST respond with error 455 (Method Not Valid In This State). The reasons to support changing transport parameters include allowing application layer mobility and flexibility to utilize the best available transport as it becomes available. If a client receives a 455 when trying to change transport parameters while the server is in Play state, it MAY try to put the server in ready state using PAUSE, before issuing the SETUP request again. If that also fails the changing of transport parameters will require that the client performs a TEARDOWN of the affected media and then to set it up again. In aggregated session avoiding tearing down all the media at the same time will avoid the creation of a new session.

All transport parameters MAY be changed. However, the primary usage expected is to either change the transport protocol completely, like switching from Interleaved TCP mode to UDP or vice versa, or to change the delivery address.

In a SETUP response for a request to change the transport parameters while in Play state, the server MUST include the Range to indicate at what point the new transport parameters will be used. Further, if RTP is used for delivery, the server MUST also include the RTP-Info header to indicate at what timestamp and RTP sequence number the change will take place. If both RTP-Info and Range are included in the response the "rtp_time" parameter and start point in the Range header MUST be for the corresponding time, i.e. be used in the same way as for PLAY to ensure the correct synchronization information is available.

If the transport parameters change while in Play state results in a change of synchronization related information, for example changing RTP SSRC, the server MUST provide in the SETUP response the necessary synchronization information. However, the server is RECOMMENDED to

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1361
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

avoid changing the synchronization information if possible.

13.4. PLAY

This section describes the usage of the PLAY method in general, for aggregated sessions, and in different usage scenarios.

13.4.1. General Usage

The PLAY method tells the server to start sending data via the mechanism specified in SETUP and which part of the media should be played out. PLAY requests are valid when the session is in Ready or Play states. A PLAY request MUST include a Session header to indicate which session the request applies to.

Upon receipt of the PLAY request, the server MUST position the normal play time to the beginning of the range specified in the received Range header and deliver stream data until the end of the range if given, until a new PLAY request is received, or until the end of the media is reached. If no Range header is present in the PLAY request the server SHALL play from current pause point until the end of media. The pause point defaults at session start to the beginning of the media. For media that is time-progressing and has no retention, the pause point will always be set equal to NPT "now", i.e., the current delivery point. The pause point may also be set to a particular point in the media by the PAUSE method, see Section 13.6. The pause point for media that is currently playing is equal to the current media position. For time-progressing media with time-limited retention, if the pause point represents a position that is older than what is retained by the server, the pause point will be moved to the oldest retained.

What range values are valid depends on the type of content. For content that isn't time progressing the range value is valid if the given range is part of any media within the aggregate. In other words the valid media range for the aggregate is the union of all of the media components in the aggregate. If a given range value points outside of the media, the response MUST be the 457 (Invalid Range) error code and include the Media-Range header (Section 16.29) with the valid range for the media. Except for time progressing content where the client requests a start point prior to what is retained, the start point is adjusted to the oldest retained content. For a start point that is beyond the media front edge, i.e. beyond the current value for "now", the server SHALL adjust the start value to the current front edge. The Range header's stop point value may point beyond the current media edge. In that case, the server SHALL deliver media from the requested (and possibly adjusted) start point until the provided stop point, or the end of the media is reached

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1362
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

prior to the specified stop point. Please note that if one simply wants to play from a particular start point until the end of media using an Range header with an implicit stop point is RECOMMENDED.

If a client requests to start playing at the end of media, either explicitly with a Range header or implicitly with a pause point that is at the end of media, a 457 (Invalid Range) error MUST be sent and include the Media-Range header (Section 16.29). Below is specified that the Range header also must be included, and will in the case of Ready State carry the pause point. Note that this also applies if the pause point or requested start point is at the beginning of the media and a Scale header (Section 16.44) is included with a negative value (playing backwards).

For media with random access properties a client may express its preference on which policy for start point selection the server shall use. This is done by including the Seek-Style header (Section 16.45) in the PLAY request. The Seek-Style applied will effect the content of the Range header as it will be adjusted to indicate from what point the media actually is delivered.

A client desiring to play the media from the beginning MUST send a PLAY request with a Range header pointing at the beginning, e.g. npt=0-. If a PLAY request is received without a Range header and media delivery has stopped at the end, the server SHOULD respond with a 457 "Invalid Range" error response. In that response, the current pause point MUST be included in a Range header.

All range specifiers in this specification allow for ranges with an implicit start point (e.g. "npt=-30"). When used in a PLAY request, the server treats this as a request to start or resume delivery from the current pause point, ending at the end time specified in the Range header. If the pause point is located later than the given end value, a 457 (Invalid Range) response MUST be given.

The example below will play seconds 10 through 25. It also requests the server to deliver media from the first Random Access Point prior to the indicated start point.

```
  C->S: PLAY rtsp://audio.example.com/audio RTSP/2.0
        CSeq: 835
        Session: 12345678
        Range: npt=10-25
        Seek-Style: RAP
        User-Agent: PhonyClient/1.2
```

Servers MUST include a "Range" header in any PLAY response, even if no Range header was present in the request. The response MUST use

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1363
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

the same format as the request's range header contained. If no Range header was in the request, the format used in any previous PLAY request within the session SHOULD be used. If no format has been indicated in a previous request the server MAY use any time format supported by the media and indicated in the Accept-Ranges header in the SETUP request. It is RECOMMENDED that NPT is used if supported by the media.

For any error response to a PLAY request, the server's response depends on the current session state. If the session is in Ready state, the current pause-point is returned using Range header with the pause point as the explicit start-point and an implicit stop-point. For time-progressing content where the pause-point moves with real-time due to limited retention, the current pause point is returned. For sessions in Play state, the current playout point and the remaining parts of the range request is returned. For any media with retention longer than 0 seconds the currently valid Media-Range header SHALL also be included in the response.

A PLAY response MAY include a header carrying synchronization information. As the information necessary is dependent on the media transport format, further rules specifying the header and its usage are needed. For RTP the RTP-Info header is specified, see Section 16.43, and used in the following example.

Here is a simple example for a single audio stream where the client requests the media starting from 3.52 seconds and to the end. The server sends a 200 OK response with the actual play time which is 10 ms prior (3.51) and the RTP-Info header that contains the necessary parameters for the RTP stack.

```
C->S: PLAY rtsp://example.com/audio RTSP/2.0
      CSeq: 836
      Session: 12345678
      Range: npt=3.52-
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 836
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Server: PhonyServer/1.0
      Range: npt=3.51-324.39
      Seek-Style: First-Prior
      RTP-Info:url="rtsp://example.com/audio"
         ssrc=0D12F123:seq=14783;rtptime=2345962545

S->C: RTP Packet TS=2345962545 => NPT=3.51
      Media duration=0.16 seconds
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1364
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The server replies with the actual start point that will be delivered. This may differ from the requested range if alignment of the requested range to valid frame boundaries is required for the media source. Note that some media streams in an aggregate may need to be delivered from even earlier points. Also, some media formats have a very long duration per individual data unit, therefore it might be necessary for the client to parse the data unit, and select where to start. The server SHALL also indicate which policy it uses for selecting the actual start point by including a Seek-Style header.

In the following example the client receives the first media packet that stretches all the way up and past the requested playtime. Thus, it is the client's decision whether to render to the user the time between 3.52 and 7.05, or to skip it. In most cases it is probably most suitable not to render that time period.

```
C->S: PLAY rtsp://example.com/audio RTSP/2.0
      CSeq: 836
      Session: 12345678
      Range: npt=7.05-
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 836
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Server: PhonyServer/1.0
      Range: npt=3.52-
      Seek-Style: First-Prior
      RTP-Info:url="rtsp://example.com/audio"
         ssrc=0D12F123:seq=14783;rtptime=2345962545

S->C: RTP Packet TS=2345962545 => NPT=3.52
      Duration=4.15 seconds
```

After playing the desired range, the presentation does NOT transition to the Ready state, media delivery simply stops. A PAUSE request MUST be issued before the stream enters the Ready state. A PLAY request while the stream is still in the Play state is legal, and can be issued without an intervening PAUSE request. Such a request MUST replace the current PLAY action with the new one requested, i.e. being handle the same as the request was received in Ready state. In the case the range in Range header has a implicit start time (-endtime), the server MUST continue to play from where it currently was until the specified end point. This is useful to change end at another point than in the previous request.

The following example plays the whole presentation starting at SMPTE

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1365
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

time code 0:10:20 until the end of the clip. Note: The RTP-Info headers has been broken into several lines, where following lines start with whitespace as allowed by the syntax.

```
C->S: PLAY rtsp://audio.example.com/twister.en RTSP/2.0
      CSeq: 833
      Session: 12345678
      Range: smpte=0:10:20-
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 833
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Session: 12345678
      Server: PhonyServer/1.0
      Range: smpte=0:10:22-0:15:45
      Seek-Style: Next
      RTP-Info:url="rtsp://example.com/twister.en"
         ssrc=0D12F123:seq=14783;rtptime=2345962545
```

For playing back a recording of a live presentation, it may be desirable to use clock units:

```
C->S: PLAY rtsp://audio.example.com/meeting.en RTSP/2.0
      CSeq: 835
      Session: 12345678
      Range: clock=19961108T142300Z-19961108T143520Z
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 835
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Session: 12345678
      Server: PhonyServer/1.0
      Range: clock=19961108T142300Z-19961108T143520Z
      Seek-Style: Next
      RTP-Info:url="rtsp://example.com/meeting.en"
         ssrc=0D12F123:seq=53745;rtptime=484589019
```

13.4.2. Aggregated Sessions

PLAY requests can operate on sessions controlling a single media and on aggregated sessions controlling multiple media.

In an aggregated session the PLAY request MUST contain an aggregated control URI. A server MUST respond with error 460 (Only Aggregate Operation Allowed) if the client PLAY Request-URI is for a single media. The media in an aggregate MUST be played in sync. If a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1366
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

client wants individual control of the media, it needs to use separate RTSP sessions for each media.

For aggregated sessions where the initial SETUP request (creating a session) is followed by one or more additional SETUP requests, a PLAY request MAY be pipelined after those additional SETUP requests without awaiting their responses. This procedure can reduce the delay from start of session establishment until media play-out has started with one round trip time. However, a client needs to be aware that using this procedure will result in the playout of the server state established at the time of processing the PLAY, i.e., after the processing of all the requests prior to the PLAY request in the pipeline. This state may not be the intended one due to failure of any of the prior requests. A client can easily determine this based on the responses from those requests. In case of failure, the client can halt the media playout using PAUSE and try to establish the intended state again before issuing another PLAY request.

13.4.3. Updating current PLAY Requests

Clients can issue PLAY requests while the stream is in Play state and thus updating their request.

The important difference compared to a PLAY request in Ready state is the handling of the current play point and how the Range header in request is constructed. The session is actively playing media and the play point will be moving, making the exact time a request will take action is hard to predict. Depending on how the PLAY header appears two different cases exist: total replacement or continuation. A total replacement is signaled by having the first range specification have an explicit start value, e.g. npt=45- or npt=45-60, in which case the server stops playout at the current playout point and then starts delivering media according to the Range header. This is equivalent to having the client first send a PAUSE and then a new play request that isn't based on the pause point. In the case of continuation the first range specifier has an implicit start point and a explicit stop value (Z), e.g. npt=-60, which indicate that it MUST convert the range specifier being played prior to this PLAY request (X to Y) into (X to Z) and continue as this was the request originally played. If the current delivery point is beyond the stop point, the server SHALL immediately pause delivery. As the request has been completed successfully it shall be responded with 200 OK. A PLAY_NOTIFY with end-of-stream is also sent to indicate the actual stop point. The pause point is set to the requested stop point.

Following is an example of this behavior: The server has received requests to play ranges 10 to 15. If the new PLAY request arrives at

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1367
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

the server 4 seconds after the previous one, it will take effect while the server still plays the first range (10-15). The server changes the current play to continue to 25 seconds, i.e. the equivalent single request would be PLAY with range: npt=10-25.

```
C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 834
      Session: 12345678
      Range: npt=10-15
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 834
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Session: 12345678
      Server: PhonyServer/1.0
      Range: npt=10-15
      Seek-Style: Next
      RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
               ssrc=0D12F123:seq=5712;rtptime=934207921,
               url="rtsp://example.com/fizzle/videotrack"
               ssrc=789DAF12:seq=57654;rtptime=2792482193
      Session: 12345678

C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 835
      Session: 12345678
      Range: npt=-25
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 835
      Date: Thu, 23 Jan 1997 15:35:09 GMT
      Session: 12345678
      Server: PhonyServer/1.0
      Range: npt=14-25
      Seek-Style: Next
      RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
               ssrc=0D12F123:seq=5712;rtptime=934239921,
               url="rtsp://example.com/fizzle/videotrack"
               ssrc=789DAF12:seq=57654;rtptime=2792842193
```

A common use of a PLAY request while in Play state is changing the scale of the media, i.e., entering or leaving from fast forward or fast rewind. The client can issue an updating PLAY request that is either a continuation or a complete replacement, as discussed above this section. We give an example of a client that is requesting a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1368
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

fast forward without giving a stop point and the change from fast forward to regular playout (scale = 1).

```
  C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
        CSeq: 2034
        Session: 12345678
        Range: npt=now-
        Scale: 2.0
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 2034
        Date: Thu, 23 Jan 1997 15:35:06 GMT
        Session: 12345678
        Server: PhonyServer/1.0
        Range: npt=2:17:21.394-
        Seek-Style: Next
        RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
                 ssrc=0D12F123:seq=5712;rtptime=934207921,
                 url="rtsp://example.com/fizzle/videotrack"
                 ssrc=789DAF12:seq=57654;rtptime=2792482193
        Session: 12345678

[playing in fast forward and now returning to scale = 1]

  C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
        CSeq: 2035
        Session: 12345678
        Range: npt=now-
        Scale: 1.0
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 2035
        Date: Thu, 23 Jan 1997 15:35:09 GMT
        Session: 12345678
        Server: PhonyServer/1.0
        Range: npt=2:19:32.144-
        Seek-Style: Next
        RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
                 ssrc=0D12F123:seq=5712;rtptime=934239921,
                 url="rtsp://example.com/fizzle/videotrack"
                 ssrc=789DAF12:seq=57654;rtptime=2792842193
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1369
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

13.4.4. Playing On-Demand Media

On-demand media is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access property is set to Random Access;
- Content Modifications set to Immutable;
- Retention set to Unlimited or Time-Limited.

Playing on-demand media follows the general usage as described in Section 13.4.1.

13.4.5. Playing Dynamic On-Demand Media

Dynamic on-demand media is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- RandomAccess set to Random-Access;
- Content Modifications set to Dynamic;
- Retention set to Unlimited or Time-Limited.

Playing on-demand media follows the general usage as described in Section 13.4.1 as long as the media has not been changed.

There are two ways for the client to be informed about changes of media resources in Play state. The client will receive a PLAY_NOTIFY request with Notify-Reason header set to media-properties-update (see Section 13.5.2. The client can use the value of the Media-Range to decide further actions, if the Media-Range header is present in the PLAY_NOTIFY request. The second way is that the client issues a GET_PARAMETER request without a body but including a Media-Range header. The 200 OK response MUST include the current Media-Range header (see Section 16.29).

13.4.6. Playing Live Media

Live media is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to No-Seeking;
- Content Modifications set to Time-Progressing;

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1370
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Retention with Time-Duration set to 0.0.

For live media, the SETUP response 200 OK MUST include the Media-Range header (see Section 16.29).

A client MAY send PLAY requests without the Range header. If the request includes the Range header it MUST use a symbolic value representing "now". For NPT that range specification is "npt=now-". The server MUST include the Range header in the response and it MUST indicate an explicit time value and not a symbolic value. In other words, "npt=now-" is not a valid to use in the response. Instead the time since session start is recommended expressed as an open interval, e.g. "npt=96.23-". An absolute time value (clock) for the corresponding time MAY be given, i.e. "clock=20030213T143205Z-". The UTC clock format can only be used if client has shown support for it using the Accept-Ranges header.

13.4.7. Playing Live with Recording

Certain media server may offer recording services of live sessions to their clients. This recording would normally be from the beginning of the media session. Clients can randomly access the media between now and the beginning of the media session. This live media with recording is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to Random-Access;

- Content Modifications set to Time-Progressing;

- Retention set to Time-limited or Unlimited

The SETUP response 200 OK MUST include the Media-Range header (see Section 16.29) for this type of media. For live media with recording, the Range header indicates the current delivery point in the media and the Media-Range header indicates the currently available media window around the current time. This window can cover recorded content in the past (seen from current time in the media) or recorded content in the future (seen from current time in the media). The server adjusts the delivery point to the requested border of the window, if the client requests a delivery point that is located outside the recording windows, e.g., if requested to far in the past, the server selects the oldest range in the recording. The considerations in Section 13.5.3 apply, if a client requests delivery with Scale (Section 16.44) values other than 1.0 (Normal playback rate) while delivering live media with recording.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1371
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

13.4.8. Playing Live with Time-Shift

Certain media server may offer time-shift services to their clients. This time shift records a fixed interval in the past, i.e., a sliding window recording mechanism, but not past this interval. Clients can randomly access the media between now and the interval. This live media with recording is indicated by the content of the Media-Properties header in the SETUP response by (see also Section 16.28):

- Random-Access set to Random-Access;
- Content Modifications set to Time-Progressing;
- Retention set to Time-Duration and a value indicating the recording interval (>0).

The SETUP response 200 OK MUST include the Media-Range header (see Section 16.29) for this type of media. For live media with recording the Range header indicates the current time in the media and the Media Range indicates a window around the current time. This window can cover recorded content in the past (seen from current time in the media) or recorded content in the future (seen from current time in the media). The server adjusts the play point to the requested border of the window, if the client requests a play point that is located outside the recording windows, e.g., if requested too far in the past, the server selects the oldest range in the recording. The considerations in Section 13.5.3 apply, if a client requests delivery using a Scale (Section 16.44) value other than 1.0 (Normal playback rate) while delivering live media with time-shift.

13.5. PLAY_NOTIFY

The PLAY_NOTIFY method is issued by a server to inform a client about an asynchronous event for a session in Play state. The Session header MUST be presented in a PLAY_NOTIFY request and indicates the scope of the request. Sending of PLAY_NOTIFY requests requires a persistent connection between server and client, otherwise there is no way for the server to send this request method to the client.

PLAY_NOTIFY requests have an end-to-end (i.e. server to client) scope, as they carry the Session header, and apply only to the given session. The client SHOULD immediately return a response to the server.

PLAY_NOTIFY requests MAY be used with a message body, depending on the value of the Notify-Reason header. It is described in the particular section for each Notify-Reason if a message body is used. However, currently there is no Notify-Reason that allows using a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1372
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

message body. In this case, there is a need to obey some limitations when adding new Notify-Reasons that intend to use a message body: the server can send any type of message body, but it is not ensured that the client can understand the received message body. This is related to DESCRIBE (see Section 13.2), but in this particular case the client can state its acceptable message bodies by using the Accept header. In the case of PLAY_NOTIFY, the server does not know which message bodies are understood by the client.

The Notify-Reason header (see Section 16.31) specifies the reason why the server sends the PLAY_NOTIFY request. This is extensible and new reasons MAY be added in the future. In case the client does not understand the reason for the notification it MUST respond with an 465 (Notification Reason Unknown) (Section 15.4.30) error code. Servers can send PLAY_NOTIFY with these types:

- end-of-stream (see Section 13.5.1);
- media-properties-update (see Section 13.5.2);
- scale-change (see Section 13.5.3).

13.5.1. End-of-Stream

A PLAY_NOTIFY request with Notify-Reason header set to end-of-stream indicates the completion or near completion of the PLAY request and the ending delivery of the media stream(s). The request MUST NOT be issued unless the server is in the Play state. The end of the media stream delivery notification may be used to indicate either a successful completion of the PLAY request currently being served, or to indicate some error resulting in failure to complete the request. The Request-Status header (Section 16.40) MUST be included to indicate which request the notification is for and its completion status. The message response status codes (Section 8.1.1) are used to indicate how the PLAY request concluded. The sender of a PLAY_NOTIFY can issue an updated PLAY_NOTIFY, in the case of a PLAY_NOTIFY sent with wrong information. For instance, a PLAY_NOTIFY was issued before reaching the end-of-stream, but some error occurred resulting in that the previously sent PLAY_NOTIFY contained a wrong time when the stream will end. In this case a new PLAY_NOTIFY MUST be sent including the correct status for the completion and all additional information.

PLAY_NOTIFY requests with Notify-Reason header set to end-of-stream MUST include a Range header and the Scale header if the scale value is not 1. The Range header indicates the point in the stream or streams where delivery is ending with the timescale that was used by the server in the PLAY response for the request being fulfilled. The

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1373
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

server MUST NOT use the "now" constant in the Range header; it MUST use the actual numeric end position in the proper timescale. When end-of-stream notifications are issued prior to having sent the last media packets, this is evident as the end time in the Range header is beyond the current time in the media being received by the client, e.g., npt=-15, if npt is currently at 14.2 seconds. The Scale header is to be included so that it is evident if the media time scale is moving backwards and/or have a non-default pace. The end-of-stream notification does not prevent the client from sending a new PLAY request.

If RTP is used as media transport, a RTP-Info header MUST be included, and the RTP-Info header MUST indicate the last sequence number in the seq parameter.

A PLAY_NOTIFY request with Notify-Reason header set to end-of-stream MUST NOT carry a message body.

This example request notifies the client about a future end-of-stream event:

```
S->C: PLAY_NOTIFY rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 854
      Notify-Reason: end-of-stream
      Request-Status: cseq=853 status=200 reason="OK"
      Range: npt=-145
      RTP-Info:url="rtsp://example.com/audio"
         ssrc=0D12F123:seq=14783;rtptime=2345962545
      Session: uZ3ci0K+Ld-M
      Date: Mon, 08 Mar 2010 13:37:16 GMT

C->S: RTSP/2.0 200 OK
      CSeq: 854
      User-Agent: PhonyClient/1.2
      Session: uZ3ci0K+Ld-M
```

13.5.2. Media-Properties-Update

A PLAY_NOTIFY request with Notify-Reason header set to media-properties-update indicates an update of the media properties for the given session (see Section 16.28) and/or the available media range that can be played as indicated by Media-Range (Section 16.29). PLAY_NOTIFY requests with Notify-Reason header set to media-properties-update MUST include a Media-Properties and Date header and SHOULD include a Media-Range header.

This notification MUST be sent for media that are time-progressing every time an event happens that changes the basis for making

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1374
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

estimates on how the media range progress. In addition it is RECOMMENDED that the server sends these notifications every 5 minutes for time-progressing content to ensure the long-term stability of the client estimation and allowing for clock skew detection by the client. Requests for the just mentioned reasons MUST include Media-Range header to provide current Media duration and the Range header to indicate the current playing point and any remaining parts of the requested range.

> The recommendation for sending updates every 5 minutes is due to any clock skew issues. In 5 minutes the clock skew should not become too significant as this is not used for media playback and synchronization, only for determining which content is available to the user.

A PLAY_NOTIFY request with Notify-Reason header set to media-properties-update MUST NOT carry a message body.

```
S->C: PLAY_NOTIFY rtsp://example.com/fizzle/foo RTSP/2.0
      Date: Tue, 14 Apr 2008 15:48:06 GMT
      CSeq: 854
      Notify-Reason: media-properties-update
      Session: uZ3ci0K+Ld-M
      Media-Properties: Time-Progressing,
            Time-Limited=20080415T153919.36Z, Random-Access=5.0
      Media-Range: npt=0-1:37:21.394
      Range: npt=1:15:49.873-

C->S: RTSP/2.0 200 OK
      CSeq: 854
      User-Agent: PhonyClient/1.2
      Session: uZ3ci0K+Ld-M
```

13.5.3. Scale-Change

The server may be forced to change the rate, when a client request delivery using a Scale (Section 16.44) value other than 1.0 (normal playback rate). For time progressing media with some retention, i.e. the server stores already sent content, a client requesting to play with Scale values larger than 1 may catch up with the front end of the media. The server will then be unable to continue to provide with content at Scale larger than 1 as content is only made available by the server at Scale=1. Another case is when Scale < 1 and the media retention is time-duration limited. In this case the delivery point can reach the oldest media unit available, and further playback at this scale becomes impossible as there will be no media available. To avoid having the client lose any media, the scale will need to be adjusted to the same rate at which the media is removed from the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1375
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

storage buffer, commonly Scale = 1.0.

Another case is when the content itself consists of spliced pieces or is dynamically updated. In these cases the server may be required to change from one supported scale value (different than Scale=1.0) to another. In this case the server will pick the closest value and inform the client of what it has picked. In these case the media properties will also be sent updating the supported Scale values. This enables a client to adjust the used Scale value.

To minimize impact on playback in any of the above cases the server MUST modify the playback properties and set Scale to a supportable value and continue delivery the media. When doing this modification it MUST send a PLAY_NOTIFY message with the Notify-Reason header set to "scale-change". The request MUST contain a Range header with the media time where the change took effect, a Scale header with the new value in use, Session header with the ID for the session it applies to and a Date header with the server wallclock time of the change. For time progressing content also the Media-Range and the Media-Properties at this point in time MUST be included. The Media-Properties header MUST be included if the scale change was due to the content changing what scale values that is supported.

For media streams being delivered using RTP also a RTP-Info header MUST be included. It MUST contain the rtptime parameter with a value corresponding to the point of change in that media and optionally also the sequence number.

A PLAY_NOTIFY request with Notify-Reason header set to "Scale-Change" MUST NOT carry a message body.

```
S->C: PLAY_NOTIFY rtsp://example.com/fizzle/foo RTSP/2.0
      Date: Tue, 14 Apr 2008 15:48:06 GMT
      CSeq: 854
      Notify-Reason: scale-change
      Session: uZ3ci0K+Ld-M
      Media-Properties: Time-Progressing,
            Time-Limited=20080415T153919.36Z, Random-Access=5.0
      Media-Range: npt=0-1:37:21.394
      Range: npt=1:37:21.394-
      Scale: 1
      RTP-Info: url="rtsp://example.com/fizzle/foo/audio"
          ssrc=0D12F123:rtptime=2345962545

C->S: RTSP/2.0 200 OK
      CSeq: 854
      User-Agent: PhonyClient/1.2
      Session: uZ3ci0K+Ld-M
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1376
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

13.6. PAUSE

The PAUSE request causes the stream delivery to immediately be interrupted (halted). A PAUSE request MUST be done either with the aggregated control URI for aggregated sessions, resulting in all media being halted, or the media URI for non-aggregated sessions. Any attempt to do muting of a single media with an PAUSE request in an aggregated session MUST be responded to with error 460 (Only Aggregate Operation Allowed). After resuming playback, synchronization of the tracks MUST be maintained. Any server resources are kept, though servers MAY close the session and free resources after being paused for the duration specified with the timeout parameter of the Session header in the SETUP message.

Example:

```
C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 834
      Session: 12345678
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 834
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Range: npt=45.76-75.00
```

The PAUSE request causes stream delivery to be interrupted immediately on receipt of the message and the pause point is set to the current point in the presentation. That pause point in the media stream needs to be maintained. A subsequent PLAY request without Range header resume from the pause point and play until media end.

The pause point after any PAUSE request MUST be returned to the client by adding a Range header with what remains unplayed of the PLAY request's range. For media with random access properties, if one desires to resume playing a ranged request, one simply includes the Range header from the PAUSE response and include the Seek-Style header with the Next policy in the PLAY request. For media that is time-progressing and has retention duration=0 the follow-up PLAY request to start media delivery again, will need to use "npt=now-" and not the answer given in the response to PAUSE.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1377
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   C->S: PLAY rtsp://example.com/fizzle/foo RTSP/2.0
         CSeq: 834
         Session: 12345678
         Range: npt=10-30
         User-Agent: PhonyClient/1.2

   S->C: RTSP/2.0 200 OK
         CSeq: 834
         Date: Thu, 23 Jan 1997 15:35:06 GMT
         Server: PhonyServer/1.0
         Range: npt=10-30
         Seek-Style: First-Prior
         RTP-Info:url="rtsp://example.com/fizzle/audiotrack"
                  ssrc=0D12F123:seq=5712;rtptime=934207921,
                  url="rtsp://example.com/fizzle/videotrack"
                  ssrc=4FAD8726:seq=57654;rtptime=2792482193
         Session: 12345678
```

After 11 seconds, i.e. at 21 seconds into the presentation:

```
   C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
         CSeq: 835
         Session: 12345678
         User-Agent: PhonyClient/1.2

   S->C: RTSP/2.0 200 OK
         CSeq: 835
         Date: 23 Jan 1997 15:35:09 GMT
         Server: PhonyServer/1.0
         Range: npt=21-30
         Session: 12345678
```

If a client issues a PAUSE request and the server acknowledges and enters the Ready state, the proper server response, if the player issues another PAUSE, is still 200 OK. The 200 OK response MUST include the Range header with the current pause point. See examples below:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1378
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 834
      Session: 12345678
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 834
      Session: 12345678
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Range: npt=45.76-98.36

C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 835
      Session: 12345678
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 835
      Session: 12345678
      Date: 23 Jan 1997 15:35:07 GMT
      Range: npt=45.76-98.36
```

13.7. TEARDOWN

13.7.1. Client to Server

The TEARDOWN client to server request stops the stream delivery for the given URI, freeing the resources associated with it. A TEARDOWN request MAY be performed on either an aggregated or a media control URI. However, some restrictions apply depending on the current state. The TEARDOWN request MUST contain a Session header indicating what session the request applies to.

A TEARDOWN using the aggregated control URI or the media URI in a session under non-aggregated control (single media session) MAY be done in any state (Ready and Play). A successful request MUST result in that media delivery being immediately halted and the session state being destroyed. This MUST be indicated through the lack of a Session header in the response.

A TEARDOWN using a media URI in an aggregated session MAY only be done in Ready state. Such a request only removes the indicated media stream and associated resources from the session. This may result in that a session returns to non-aggregated control, due to that it only contains a single media after the requests completion. A session that will exist after the processing of the TEARDOWN request MUST in the response to that TEARDOWN request contain a Session header. Thus the presence of the Session header indicates to the receiver of the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1379
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

response if the session is still existing or has been removed.

Example:

```
  C->S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/2.0
        CSeq: 892
        Session: 12345678
        User-Agent: PhonyClient/1.2

  S->C: RTSP/2.0 200 OK
        CSeq: 892
        Server: PhonyServer/1.0
```

13.7.2. Server to Client

The server can send TEARDOWN requests in the server to client direction to indicate that the server has been forced to terminate the ongoing session. This may happen for several reasons, such as server maintenance without available backup, or that the session has been inactive for extended periods of time. The reason is provided in the Terminate-Reason header (Section 16.50).

When a RTSP client has maintained a RTSP session that otherwise is inactive for an extended period of time the server may reclaim the resources. That is done by issuing a TEARDOWN request with the Terminate-Reason set to "Session-Timeout". This MAY be done when the client has been inactive in the RTSP session for more than one Session Timeout period (Section 16.47). However, the server is RECOMMENDED to not perform this operation until an extended period of inactivity has passed. The time period is considered extended when it is 10 times the Session Timeout period. Consideration of the application of the server and its content should be performed when configuring what is considered as extended periods of time.

In case the server needs to stop providing service to the established sessions and their is no server to point at in a REDIRECT request TEARDOWN shall be used to terminate the session. This method can also be used when non-recoverable internal errors have happened and the server has no other option then to terminate the sessions.

The TEARDOWN request MUST be only done on the session aggregate control URI (i.e., it is not allowed to terminate individual media streams) and MUST include the following headers; Session and Terminate-Reason headers. The request only applies to the session identified in the Session header. The server may include a message to the client's user with the "user-msg" parameter.

The TEARDOWN request may alternatively be done on the wild card URI *

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1380
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

and without any session header. The scope of such a request is limited to the next-hop (i.e. the RTSP agent in direct communication with the server) and applies, as well, to the control connection between the next-hop RTSP agent and the server. This request indicates that all sessions and pending requests being managed via the control connection are terminated. Any intervening proxies SHOULD do all of the following in the order listed:

1. respond to the TEARDOWN request

2. disconnect the control channel from the requesting server

3. pass the TEARDOWN request to each applicable client (typically those clients with an active session or an unanswered request)

Note: The proxy is responsible for accepting TEARDOWN responses from its clients; these responses MUST NOT be passed on to either the original server or the target server in the redirect.

13.8. GET_PARAMETER

The GET_PARAMETER request retrieves the value of any specified parameter or parameters for a presentation or stream specified in the URI. If the Session header is present in a request, the value of a parameter MUST be retrieved in the specified session context. There are two ways of specifying the parameters to be retrieved. The first is by including headers which have been defined such that you can use them for this purpose. Headers for this purpose should allow empty, or stripped value parts to avoid having to specify bogus data when indicating the desire to retrieve a value. The successful completion of the request should also be evident from any filled out values in the response. The Media-Range header (Section 16.29) is one such header. The other way is to specify a message body that lists the parameter(s) that are desired to be retrieved. The Content-Type header (Section 16.18) is used to specify which format the message body has.

The headers that MAY be used for retrieving their current value using GET_PARAMETER are:

- Accept-Ranges
- Media-Range
- Media-Properties
- Range

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1381
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

o RTP-Info

The method MAY also be used without a message body or any header that request parameters for keep-alive purpose. Any request that is successful, i.e., a 200 OK response is received, then the keep-alive timer has been updated. Any non-required header present in such a request may or may not been processed. Normally the presence of filled out values in the header will be indication that the header has been processed. However, for cases when this is difficult to determine, it is recommended to use a feature-tag and the Require header. Due to this reason it is usually easier if any parameters to be retrieved are sent in the body, rather than using any header.

Parameters specified within the body of the message must all be understood by the request receiving agent. If one or more parameters are not understood a 451 (Parameter Not Understood) MUST be sent including a body listing these parameters that weren't understood. If all parameters are understood their values are filled in and returned in the response message body.

Example:

```
S->C: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 431
      User-Agent: PhonyClient/1.2
      Session: 12345678
      Content-Length: 26
      Content-Type: text/parameters

      packets_received
      jitter

C->S: RTSP/2.0 200 OK
      CSeq: 431
      Session: 12345678
      Server: PhonyServer/1.1
      Date: Mon, 08 Mar 2010 13:43:23 GMT
      Content-Length: 38
      Content-Type: text/parameters

      packets_received: 10
      jitter: 0.3838
```

13.9. SET_PARAMETER

This method requests to set the value of a parameter or a set of parameters for a presentation or stream specified by the URI. The method MAY also be used without a message body. It is the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1382
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RECOMMENDED method to be used in a request sent for the sole purpose of updating the keep-alive timer. If this request is successful, i.e. a 200 OK response is received, then the keep-alive timer has been updated. Any non-required header present in such a request may or may not been processed. To allow a client to determine if any such header has been processed, it is necessary to use a feature tag and the Require header. Due to this reason it is RECOMMENDED that any parameters are sent in the body, rather than using any header.

A request is RECOMMENDED to only contain a single parameter to allow the client to determine why a particular request failed. If the request contains several parameters, the server MUST only act on the request if all of the parameters can be set successfully. A server MUST allow a parameter to be set repeatedly to the same value, but it MAY disallow changing parameter values. If the receiver of the request does not understand or cannot locate a parameter, error 451 (Parameter Not Understood) MUST be used. In the case a parameter is not allowed to change, the error code is 458 (Parameter Is Read-Only). The response body MUST contain only the parameters that have errors. Otherwise no body MUST be returned.

Note: transport parameters for the media stream MUST only be set with the SETUP command.

> Restricting setting transport parameters to SETUP is for the benefit of firewalls.
>
> The parameters are split in a fine-grained fashion so that there can be more meaningful error indications. However, it may make sense to allow the setting of several parameters if an atomic setting is desirable. Imagine device control where the client does not want the camera to pan unless it can also tilt to the right angle at the same time.

Example:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1383
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
     C->S: SET_PARAMETER rtsp://example.com/fizzle/foo RTSP/2.0
           CSeq: 421
           User-Agent: PhonyClient/1.2
           Session: iixT43KLc
           Date: Mon, 08 Mar 2010 14:45:04 GMT
           Content-length: 20
           Content-type: text/parameters

           barparam: barstuff

     S->C: RTSP/2.0 451 Parameter Not Understood
           CSeq: 421
           Session: iixT43KLc
           Server: PhonyServer/1.0
           Date: Mon, 08 Mar 2010 14:44:56 GMT
           Content-length: 10
           Content-type: text/parameters

           barparam: barstuff
```

13.10. REDIRECT

The REDIRECT method is issued by a server to inform a client that the service provided will be terminated and where a corresponding service can be provided instead. This may happen for different reasons. One is that the server is being administrated such that it must stop providing service. Thus the client is required to connect to another server location to access the resource indicated by the Request-URI.

The REDIRECT request SHALL contain a Terminate-Reason header (Section 16.50) to inform the client of the reason for the request. Additional parameters related to the reason may also be included. The intention here is to allow a server administrator to do a controlled shutdown of the RTSP server. That requires sufficient time to inform all entities having associated state with the server and for them to perform a controlled migration from this server to a fall back server.

A REDIRECT request with a Session header has end-to-end (i.e. server to client) scope and applies only to the given session. Any intervening proxies SHOULD NOT disconnect the control channel while there are other remaining end-to-end sessions. The REQUIRED Location header MUST contain a complete absolute URI pointing to the resource to which the client SHOULD reconnect. Specifically, the Location MUST NOT contain just the host and port. A client may receive a REDIRECT request with a Session header, if and only if, an end-to-end session has been established.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1384
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

A client may receive a REDIRECT request without a Session header at any time when it has communication or a connection established with a server. The scope of such a request is limited to the next-hop (i.e. the RTSP agent in direct communication with the server) and applies to all sessions controlled, as well as the control connection between the next-hop RTSP agent and the server. A REDIRECT request without a Session header indicates that all sessions and pending requests being managed via the control connection MUST be redirected. The REQUIRED Location header, if included in such a request, SHOULD contain an absolute URI with only the host address and the OPTIONAL port number of the server to which the RTSP agent SHOULD reconnect. Any intervening proxies SHOULD do all of the following in the order listed:

1. respond to the REDIRECT request

2. disconnect the control channel from the requesting server

3. connect to the server at the given host address

4. pass the REDIRECT request to each applicable client (typically those clients with an active session or an unanswered request)

   Note: The proxy is responsible for accepting REDIRECT responses from its clients; these responses MUST NOT be passed on to either the original server or the redirected server.

When the server lacks any alternative server and needs to terminate a session or all sessions the TEARDOWN request SHALL be used instead.

When no Terminate-Reason "time" parameter are included in a REDIRECT request, the client SHALL perform the redirection immediately and return a response to the server. The server shall consider the session as terminated and can free any associated state after it receives the successful (2xx) response. The server MAY close the signaling connection upon receiving the response and the client SHOULD close the signaling connection after sending the 2xx response. The exception to this is when the client has several sessions on the server being managed by the given signaling connection. In this case, the client SHOULD close the connection when it has received and responded to REDIRECT requests for all the sessions managed by the signaling connection.

The Terminate-Reason header "time" parameter MAY be used to indicate the wallclock time by when the redirection MUST have take place. To allow a client to determine that redirect time without being time synchronized with the server, the server MUST include a Date header in the request. The client should have before the redirection time-

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1385
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

line terminated the session and close the control connection. The server MAY simple cease to provide service when the deadline time has been reached, or it may issue TEARDOWN requests to the remaining sessions.

If the REDIRECT request times out following the rules in Section 10.4 the server MAY terminate the session or transport connection that would be redirected by the request. This is a safeguard against misbehaving clients that refuse to respond to a REDIRECT request. That should not provide any benefit.

After a REDIRECT request has been processed, a client that wants to continue to send or receive media for the resource identified by the Request-URI will have to establish a new session with the designated host. If the URI given in the Location header is a valid resource URI, a client SHOULD issue a DESCRIBE request for the URI.

> Note: The media resource indicated by the Location header can be identical, slightly different or totally different. This is the reason why a new DESCRIBE request SHOULD be issued.

If the Location header contains only a host address, the client MAY assume that the media on the new server is identical to the media on the old server, i.e. all media configuration information from the old session is still valid except for the host address. However, the usage of conditional SETUP using MTag identifiers are RECOMMENDED to verify the assumption.

This example request redirects traffic for this session to the new server at the given absolute time:

```
S->C: REDIRECT rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 732
      Location: rtsp://s2.example.com:8001
      Terminate-Reason: Server-Admin ;time=19960213T143205Z
      Session: uZ3ci0K+Ld-M
      Date: Thu, 13 Feb 1996 14:30:43 GMT

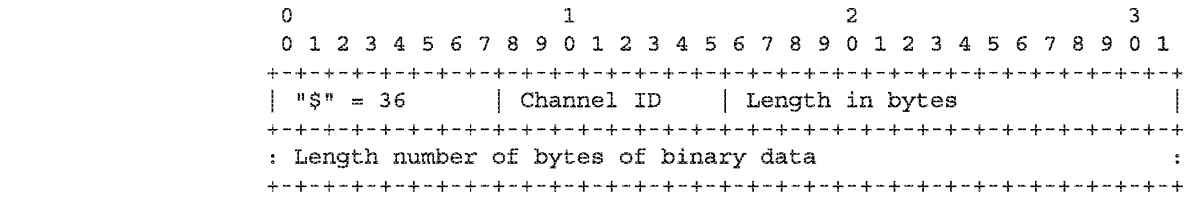
C->S: RTSP/2.0 200 OK
      CSeq: 732
      User-Agent: PhonyClient/1.2
      Session: uZ3ci0K+Ld-M
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1386
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

14. Embedded (Interleaved) Binary Data

In order to fulfill certain requirements on the network side, e.g. in conjunction with network address translators that block RTP traffic over UDP, it may be necessary to interleave RTSP messages and media stream data. This interleaving should generally be avoided unless necessary since it complicates client and server operation and imposes additional overhead. Also, head of line blocking may cause problems. Interleaved binary data SHOULD only be used if RTSP is carried over TCP. Interleaved data is not allowed inside RTSP messages.

Stream data such as RTP packets is encapsulated by an ASCII dollar sign (36 decimal), followed by a one-byte channel identifier, followed by the length of the encapsulated binary data as a binary, two-byte integer in network byte order. The stream data follows immediately afterwards, without a CRLF, but including the upper-layer protocol headers. Each $ block MUST contain exactly one upper-layer protocol data unit, e.g., one RTP packet.



The channel identifier is defined in the Transport header with the interleaved parameter (Section 16.52).

When the transport choice is RTP, RTCP messages are also interleaved by the server over the TCP connection. The usage of RTCP messages is indicated by including a interval containing a second channel in the interleaved parameter of the Transport header, see Section 16.52. If RTCP is used, packets MUST be sent on the first available channel higher than the RTP channel. The channels are bi-directional, using the same ChannelD in both directions, and therefore RTCP traffic are sent on the second channel in both directions.

> RTCP is sometime needed for synchronization when two or more streams are interleaved in such a fashion. Also, this provides a convenient way to tunnel RTP/RTCP packets through the TCP control connection when required by the network configuration and transfer them onto UDP when possible.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1387
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
C->S: SETUP rtsp://example.com/bar.file RTSP/2.0
      CSeq: 2
      Transport: RTP/AVP/TCP;unicast;interleaved=0-1
      Accept-Ranges: NPT, SMPTE, UTC
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 2
      Date: Thu, 05 Jun 1997 18:57:18 GMT
      Transport: RTP/AVP/TCP;unicast;interleaved=5-6
      Session: 12345678
      Accept-Ranges: NPT
      Media-Properties: Random-Access=0.2, Immutable, Unlimited

C->S: PLAY rtsp://example.com/bar.file RTSP/2.0
      CSeq: 3
      Session: 12345678
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 3
      Session: 12345678
      Date: Thu, 05 Jun 1997 18:57:19 GMT
      RTP-Info: url="rtsp://example.com/bar.file"
        ssrc=0D12F123:seq=232433;rtptime=972948234
      Range: npt=0-56.8
      Seek-Style: RAP

S->C: $005{2 byte length}{"length" bytes data, w/RTP header}
S->C: $005{2 byte length}{"length" bytes data, w/RTP header}
S->C: $006{2 byte length}{"length" bytes  RTCP packet}
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1388
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15. Status Code Definitions

Where applicable, HTTP status [H10] codes are reused. Status codes that have the same meaning are not repeated here. See Table 4 for a listing of which status codes may be returned by which requests. All error messages, 4xx and 5xx MAY return a body containing further information about the error.

15.1. Success 1xx

15.1.1. 100 Continue

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed.

15.2. Success 2xx

This class of status code indicates that the client's request was successfully received, understood, and accepted.

15.2.1. 200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request.

15.3. Redirection 3xx

The notation "3rr" indicates response codes from 300 to 399 inclusive which are meant for redirection. The response code 304 is excluded from this set, as it is not used for redirection.

Within RTSP, redirection may be used for load balancing or redirecting stream requests to a server topologically closer to the client. Mechanisms to determine topological proximity are beyond the scope of this specification.

A 3rr code MAY be used to respond to any request. It is RECOMMENDED that they are used if necessary before a session is established, i.e., in response to DESCRIBE or SETUP. However, in cases where a server is not able to send a REDIRECT request to the client, the server MAY need to resort to using 3rr responses to inform a client with an established session about the need for redirecting the session. If a 3rr response is received for a request in relation to

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1389
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

an established session, the client SHOULD send a TEARDOWN request for the session, and MAY reestablish the session using the resource indicated by the Location.

If the Location header is used in a response it MUST contain an absolute URI pointing out the media resource the client is redirected to, the URI MUST NOT only contain the host name.

15.3.1. 301 Moved Permanently

The request resource are moved permanently and resides now at the URI given by the location header. The user client SHOULD redirect automatically to the given URI. This response MUST NOT contain a message-body. The Location header MUST be included in the response.

15.3.2. 302 Found

The requested resource resides temporarily at the URI given by the Location header. The Location header MUST be included in the response. This response is intended to be used for many types of temporary redirects; e.g., load balancing. It is RECOMMENDED that the server set the reason phrase to something more meaningful than "Found" in these cases. The user client SHOULD redirect automatically to the given URI. This response MUST NOT contain a message-body.

This example shows a client being redirected to a different server:

```
C->S: SETUP rtsp://example.com/fizzle/foo RTSP/2.0
      CSeq: 2
      Transport: RTP/AVP/TCP;unicast;interleaved=0-1
      Accept-Ranges: NPT, SMPTE, UTC
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 302 Try Other Server
      CSeq: 2
      Location: rtsp://s2.example.com:8001/fizzle/foo
```

15.3.3. 303 See Other

This status code MUST NOT be used in RTSP 2.0. However, it was allowed to use in RTSP 1.0 (RFC 2326).

15.3.4. 304 Not Modified

If the client has performed a conditional DESCRIBE or SETUP (see Section 16.24) and the requested resource has not been modified, the server SHOULD send a 304 response. This response MUST NOT contain a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1390
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

message-body.

The response MUST include the following header fields:

- Date
- MTag and/or Content-Location, if the header(s) would have been sent in a 200 response to the same request.
- Expires, Cache-Control, and/or Vary, if the field-value might differ from that sent in any previous response for the same variant.

This response is independent for the DESCRIBE and SETUP requests. That is, a 304 response to DESCRIBE does NOT imply that the resource content is unchanged (only the session description) and a 304 response to SETUP does NOT imply that the resource description is unchanged. The MTag and If-Match headers may be used to link the DESCRIBE and SETUP in this manner.

15.3.5. 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses MUST only be generated by origin servers.

15.4. Client Error 4xx

15.4.1. 400 Bad Request

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications. If the request does not have a CSeq header, the server MUST NOT include a CSeq in the response.

15.4.2. 401 Unauthorized

The request requires user authentication. The response MUST include a WWW-Authenticate header (Section 16.57) field containing a challenge applicable to the requested resource. The client MAY repeat the request with a suitable Authorization header field. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the message body that was given in the response, since that message body

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1391
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

might include relevant diagnostic information. HTTP access authentication is explained in [RFC2617].

15.4.3. 402 Payment Required

This code is reserved for future use.

15.4.4. 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the server wishes to make public why the request has not been fulfilled, it SHOULD describe the reason for the refusal in the message body. If the server does not wish to make this information available to the client, the status code 404 (Not Found) can be used instead.

15.4.5. 404 Not Found

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

15.4.6. 405 Method Not Allowed

The method specified in the request is not allowed for the resource identified by the Request-URI. The response MUST include an Allow header containing a list of valid methods for the requested resource. This status code is also to be used if a request attempts to use a method not indicated during SETUP.

15.4.7. 406 Not Acceptable

The resource identified by the request is only capable of generating response message bodies which have content characteristics not acceptable according to the accept headers sent in the request.

The response SHOULD include an message body containing a list of available message body characteristics and location(s) from which the user or user agent can choose the one most appropriate. The message body format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1392
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

automatically. However, this specification does not define any standard for such automatic selection.

If the response could be unacceptable, a user agent SHOULD temporarily stop receipt of more data and query the user for a decision on further actions.

15.4.8. 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized) (Section 15.4.2), but indicates that the client must first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (Section 16.33) containing a challenge applicable to the proxy for the requested resource.

15.4.9. 408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time.

15.4.10. 410 Gone

The requested resource is no longer available at the server and the forwarding address is not known. This condition is expected to be considered permanent. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead. This response is cacheable unless indicated otherwise.

The 410 response is primarily intended to assist the task of repository maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as "gone" or to keep the mark for any length of time -- that is left to the discretion of the owner of the server.

15.4.11. 411 Length Required

The server refuses to accept the request without a defined Content-Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1393
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15.4.12. 412 Precondition Failed

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource meta information (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.

15.4.13. 413 Request Message Body Too Large

The server is refusing to process a request because the request message body is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

15.4.14. 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has used a request with long query information, when the client has descended into a URI "black hole" of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the Request-URI.

15.4.15. 415 Unsupported Media Type

The server is refusing to service the request because the message body of the request is in a format not supported by the requested resource for the requested method.

15.4.16. 451 Parameter Not Understood

The recipient of the request does not support one or more parameters contained in the request. When returning this error message the sender SHOULD return a message body containing the offending parameter(s).

15.4.17. 452 reserved

This error code was removed from RFC 2326 [RFC2326] as it is obsolete. This error code MUST NOT be used anymore.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1394
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15.4.18. 453 Not Enough Bandwidth

The request was refused because there was insufficient bandwidth. This may, for example, be the result of a resource reservation failure.

15.4.19. 454 Session Not Found

The RTSP session identifier in the Session header is missing, invalid, or has timed out.

15.4.20. 455 Method Not Valid in This State

The client or server cannot process this request in its current state. The response MUST contain an Allow header to make error recovery possible.

15.4.21. 456 Header Field Not Valid for Resource

The server could not act on a required request header. For example, if PLAY contains the Range header field but the stream does not allow seeking. This error message may also be used for specifying when the time format in Range is impossible for the resource. In that case the Accept-Ranges header MUST be returned to inform the client of which format(s) that are allowed.

15.4.22. 457 Invalid Range

The Range value given is out of bounds, e.g., beyond the end of the presentation.

15.4.23. 458 Parameter Is Read-Only

The parameter to be set by SET_PARAMETER can be read but not modified. When returning this error message the sender SHOULD return a message body containing the offending parameter(s).

15.4.24. 459 Aggregate Operation Not Allowed

The requested method may not be applied on the URI in question since it is an aggregate (presentation) URI. The method may be applied on a media URI.

15.4.25. 460 Only Aggregate Operation Allowed

The requested method may not be applied on the URI in question since it is not an aggregate control (presentation) URI. The method may be applied on the aggregate control URI.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1395
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15.4.26. 461 Unsupported Transport

The Transport field did not contain a supported transport specification.

15.4.27. 462 Destination Unreachable

The data transmission channel could not be established because the client address could not be reached. This error will most likely be the result of a client attempt to place an invalid dest_addr parameter in the Transport field.

15.4.28. 463 Destination Prohibited

The data transmission channel was not established because the server prohibited access to the client address. This error is most likely the result of a client attempt to redirect media traffic to another destination with a dest_addr parameter in the Transport header.

15.4.29. 464 Data Transport Not Ready Yet

The data transmission channel to the media destination is not yet ready for carrying data. However, the responding agent still expects that the data transmission channel will be established at some point in time. Note, however, that this may result in a permanent failure like 462 "Destination Unreachable".

An example when this error may occur is in the case a client sends a PLAY request to a server prior to ensuring that the TCP connections negotiated for carrying media data was successful established (In violation of this specification). The server would use this error code to indicate that the requested action could not be performed due to the failure of completing the connection establishment.

15.4.30. 465 Notification Reason Unknown

This indicates that the client has received a PLAY_NOTIFY (Section 13.5) with a Notify-Reason header (Section 16.31) unknown to the client.

15.4.31. 466 Key Management Error

This indicates that there has been an error in a Key Management function used in conjunction with a request. For example usage of MIKEY according to Appendix C.1.4.1 may result in this error.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1396
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15.4.32. 470 Connection Authorization Required

The secured connection attempt needs user or client authorization before proceeding. The next hops certificate is included in this response in the Accept-Credentials header.

15.4.33. 471 Connection Credentials not accepted

When performing a secure connection over multiple connections, a intermediary has refused to connect to the next hop and carry out the request due to unacceptable credentials for the used policy.

15.4.34. 472 Failure to establish secure connection

A proxy fails to establish a secure connection to the next hop RTSP agent. This is primarily caused by a fatal failure at the TLS handshake, for example due to server not accepting any cipher suits.

15.5. Server Error 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request The server SHOULD include an message body containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included message body to the user. These response codes are applicable to any request method.

15.5.1. 500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request.

15.5.2. 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

15.5.3. 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1397
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

15.5.4. 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response. The client MUST honor the length, if given in the Retry-After header.

> Note: The existence of the 503 status code does not imply that a server must use it when becoming overloaded. Some servers may wish to simply refuse the connection.

15.5.5. 504 Gateway Timeout

The server, while acting as a proxy, did not receive a timely response from the upstream server specified by the URI or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.

15.5.6. 505 RTSP Version Not Supported

The server does not support, or refuses to support, the RTSP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client other than with this error message. The response SHOULD contain an message body describing why that version is not supported and what other protocols are supported by that server.

15.5.7. 551 Option not supported

A feature-tag given in the Require or the Proxy-Require fields was not supported. The Unsupported header MUST be returned stating the feature for which there is no support.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1398
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16. Header Field Definitions

method	direction	object	acronym	Body
DESCRIBE	C -> S	P,S	DES	r
GET_PARAMETER	C -> S, S -> C	P,S	GPR	R,r
OPTIONS	C -> S, S -> C	P,S	OPT	
PAUSE	C -> S	P,S	PSE	
PLAY	C -> S	P,S	PLY	
PLAY_NOTIFY	S -> C	P,S	PNY	R
REDIRECT	S -> C	P,S	RDR	
SETUP	C -> S	S	STP	
SET_PARAMETER	C -> S, S -> C	P,S	SPR	R,r
TEARDOWN	C -> S	P,S	TRD	
	S -> C	P	TRD	

Table 8: Overview of RTSP methods, their direction, and what objects (P: presentation, S: stream) they operate on. Body notes if a method is allowed to carry body and in which direction, R = Request, r=response. Note: It is allowed for all error messages 4xx and 5xx to have a body

The general syntax for header fields is covered in Section 5.2. This section lists the full set of header fields along with notes on meaning, and usage. The syntax definition for header fields are present in Section 20.2.3. Throughout this section, we use [HX.Y] to informational refer to Section X.Y of the current HTTP/1.1 specification RFC 2616 [RFC2616]. Examples of each header field are given.

Information about header fields in relation to methods and proxy processing is summarized in Table 9, Table 10, Table 11, and Table 12.

The "where" column describes the request and response types in which the header field can be used. Values in this column are:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1399
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

R: header field may only appear in requests;

r: header field may only appear in responses;

2xx, 4xx, etc.: A numerical value or range indicates response codes with which the header field can be used;

c: header field is copied from the request to the response.

An empty entry in the "where" column indicates that the header field may be present in both requests and responses.

The "proxy" column describes the operations a proxy may perform on a header field. An empty proxy column indicates that the proxy MUST NOT do any changes to that header, all allowed operations are explicitly stated:

a: A proxy can add or concatenate the header field if not present.

m: A proxy can modify an existing header field value.

d: A proxy can delete a header field value.

r: A proxy needs to be able to read the header field, and thus this header field cannot be encrypted.

The rest of the columns relate to the presence of a header field in a method. The method names when abbreviated, are according to Table 8:

c: Conditional; requirements on the header field depend on the context of the message.

m: The header field is mandatory.

m*: The header field SHOULD be sent, but clients/servers need to be prepared to receive messages without that header field.

o: The header field is optional.

*: The header field MUST be present if the message body is not empty. See Section 16.16, Section 16.18 and Section 5.3 for details.

-: The header field is not applicable.

"Optional" means that a Client/Server MAY include the header field in a request or response. The Client/Server behavior when receiving such headers varies, for some it may ignore the header field, in

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1400
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

other case it is request to process the header. This is regulated by the method and header descriptions. Example of headers that require processing are the Require and Proxy-Require header fields discussed in Section 16.41 and Section 16.35. A "mandatory" header field MUST be present in a request, and MUST be understood by the Client/Server receiving the request. A mandatory response header field MUST be present in the response, and the header field MUST be understood by the Client/Server processing the response. "Not applicable" means that the header field MUST NOT be present in a request. If one is placed in a request by mistake, it MUST be ignored by the Client/ Server receiving the request. Similarly, a header field labeled "not applicable" for a response means that the Client/Server MUST NOT place the header field in the response, and the Client/Server MUST ignore the header field in the response.

An RTSP agent MUST ignore extension headers that are not understood.

The From and Location header fields contain an URI. If the URI contains a comma, or semicolon, the URI MUST be enclosed in double quotes ("). Any URI parameters are contained within these quotes. If the URI is not enclosed in double quotas, any semicolon- delimited parameters are header-parameters, not URI parameters.

Header	Where	Proxy	DES	OPT	STP	PLY	PSE	TRD
Accept	R		o	-	-	-	-	-
Accept-Credentials	R	rm	o	o	o	o	o	o
Accept-Encoding	R	r	o	-	-	-	-	-
Accept-Language	R	r	o	-	-	-	-	-
Accept-Ranges	R	r	-	-	m	-	-	-
Accept-Ranges	r	r	-	-	m	-	-	-
Accept-Ranges	456	r	-	-	-	m	-	-
Allow	r	am	c	c	c	-	-	-
Allow	405	am	m	m	m	m	m	m
Authorization	R		o	o	o	o	o	o

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1401
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Bandwidth	R		o	o	o	o	-	-
Blocksize	R		o	-	o	o	-	-
Cache-Control		r	o	-	o	-	-	-
Connection		ad	o	o	o	o	o	o
Connection-Credentials	470,407	ar	o	o	o	o	o	o
Content-Base	r		o	-	-	-	-	-
Content-Base	4xx,5xx		o	o	o	o	o	o
Content-Encoding	R	r	-	-	-	-	-	-
Content-Encoding	r	r	o	-	-	-	-	-
Content-Encoding	4xx,5xx	r	o	o	o	o	o	o
Content-Language	R	r	-	-	-	-	-	-
Content-Language	r	r	o	-	-	-	-	-
Content-Language	4xx,5xx	r	o	o	o	o	o	o
Content-Length	r	r	*	-	-	-	-	-
Content-Length	4xx,5xx	r	*	*	*	*	*	*
Content-Location	r	r	o	-	-	-	-	-
Content-Location	4xx,5xx	r	o	o	o	o	o	o
Content-Type	r	r	*	-	-	-	-	-
Content-Type	4xx,5xx	ar	*	*	*	*	*	*
CSeq	Rc	rm	m	m	m	m	m	m

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1402
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Date		am	o/*	o/*	o/*	o/*	o/*	o/*
Expires	r	r	o	-	-	-	-	-
From	R	r	o	o	o	o	o	o
If-Match	R	r	-	-	o	-	-	-
If-Modified-Since	R	r	o	-	o	-	-	-
If-None-Match	R	r	o	-	o	-	-	-
Last-Modified	r	r	o	-	o	-	-	-
Location	3rr		o	o	o	o	o	o

Table 9: Overview of RTSP header fields (A-L) related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.

Header	Where	Proxy	DES	OPT	STP	PLY	PSE	TRD
Media-Properties			-	-	m	m	m	-
Media-Range			-	-	m	m	m	-
MTag	r	r	o	-	o	-	-	-
Pipelined-Requests		amdr	-	o	o	o	o	o
Proxy-Authenticate	407	amr	m	m	m	m	m	m
Proxy-Authorization	R	rd	o	o	o	o	o	o
Proxy-Require	R	ar	o	o	o	o	o	o
Proxy-Require	r	r	c	c	c	c	c	c

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1403
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Proxy-Supported	R	amr	c	c	c	c	c	c
Proxy-Supported	r		c	c	c	c	c	c
Public	r	amr	-	m	-	-	-	-
Public	501	amr	m	m	m	m	m	m
Range	R		-	-	-	o	-	-
Range	r		-	-	c	m	m	-
Terminate-Reason	R	r	-	-	-	-	-	-
Referrer	R		o	o	o	o	o	o
Request-Status	R		-	-	-	-	-	-
Require	R		o	o	o	o	o	o
Retry-After	3rr,503		o	o	o	o	o	-
Retry-After	413		o	-	-	-	-	-
RTP-Info	r		-	-	c	c	-	-
Scale	R	r	-	-	-	o	-	-
Scale	r	amr	-	-	-	c	-	-
Seek-Style	R		-	-	-	o	-	-
Seek-Style	r		-	-	-	m	-	-
Server	R	r	-	o	-	-	-	o
Server	r	r	o	o	o	o	o	o
Session	R	r	-	o	o	m	m	m
Session	r	r	-	c	m	m	m	o
Speed	R	admr	-	-	-	o	-	-

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1404
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Speed	r	admr	-	-	-	c	-	-
Supported	R	amr	o	o	o	o	o	o
Supported	r	amr	c	c	c	c	c	c
Timestamp	R	admr	o	o	o	o	o	o
Timestamp	c	admr	m	m	m	m	m	m
Transport		mr	-	-	m	-	-	-
Unsupported	r		c	c	c	c	c	c
User-Agent	R		m*	m*	m*	m*	m*	m*
Vary	r		c	c	c	c	c	c
Via	R	amr	o	o	o	o	o	o
Via	c	dr	m	m	m	m	m	m
WWW-Authenticate	401		m	m	m	m	m	m

Table 10: Overview of RTSP header fields (P-W) related to methods DESCRIBE, OPTIONS, SETUP, PLAY, PAUSE, and TEARDOWN.

Header	Where	Proxy	GPR	SPR	RDR	PNY
Accept	R	arm	o	o	-	-
Accept-Credentials	R	rm	o	o	o	-
Accept-Ranges		rm	o	-	-	-
Allow	405	amr	m	m	m	-
Authorization	R		o	o	o	-
Bandwidth	R		-	o	-	-
Blocksize	R		-	o	-	-
Connection			o	o	o	o

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1405
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Cache-Control		r	o	o	-	-
Connection-Credentials	470,407	ar	o	o	o	-
Content-Base	R		o	o	-	-
Content-Base	r		o	o	-	-
Content-Base	4xx,5xx		o	o	o	o
Content-Encoding	R	r	o	o	-	-
Content-Encoding	r	r	o	o	-	-
Content-Encoding	4xx,5xx	r	o	o	o	o
Content-Language	R	r	o	o	-	-
Content-Language	r	r	o	o	-	-
Content-Language	4xx,5xx	r	o	o	o	o
Content-Length	R	r	*	*	-	-
Content-Length	r	r	*	*	-	-
Content-Length	4xx,5xx	r	*	*	*	*
Content-Location	R		o	o	-	-
Content-Location	r		o	o	-	-
Content-Location	4xx,5xx		o	o	o	o
Content-Type	R		*	*	-	-
Content-Type	r		*	*	-	-
Content-Type	4xx,5xx		*	*	*	*
CSeq	R,c	mr	m	m	m	m
Date	R	a	o	o	m	o
Date	r	am	o	o	o	o
If-Modified-Since	R	am	o	-	-	-

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1406
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

If-None-Match	R	am	o	-	-	-
From	R	r	o	o	o	-
Last-Modified	R	r	-	-	-	-
Last-Modified	r	r	o	-	-	-
Location	3rr		o	o	o	-
Location	R		-	-	m	-
Media-Properties	R	amr	o	-	-	c
Media-Properties	r	mr	c	-	-	-
Media-Range	R		o	-	-	c
Media-Range	r		c	-	-	-
Notify-Reason	R		-	-	-	m
Pipelined-Requests	R	amdr	o	o	-	-
Proxy-Authenticate	407	amr	m	m	m	-
Proxy-Authorization	R	rd	o	o	o	-
Proxy-Require	R	ar	o	o	o	-
Proxy-Require	r	r	c	c	c	-
Proxy-Supported	R	amr	c	c	c	-
Proxy-Supported	r		c	c	c	-
Public	501	admr	m	m	m	-

Table 11: Overview of RTSP header fields (A-P) related to methods GET_PARAMETER, SET_PARAMETER, REDIRECT, and PLAY_NOTIFY.

Header	Where	Proxy	GPR	SPR	RDR	PNY
Range	R		o	-	o	m
Referrer	R		o	o	o	-

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1407
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Request-Status	R		-	-	-	c
Require	R	r	o	o	o	-
Retry-After	3rr,503		o	o	-	-
Retry-After	413		o	o	-	-
RTP-Info	R	r	o	-	-	C
RTP-Info	r	r	c	-	-	-
Scale			-	-	-	c
Seek-Style			-	-	-	-
Session	R	r	o	o	o	m
Session	r	r	c	c	o	m
Server	R	r	o	o	o	o
Server	r	r	o	o	-	-
Speed			-	-	-	-
Supported	R	adrm	o	o	o	-
Supported	r	adrm	c	c	c	-
Terminate-Reason	R	r	-	-	m	-
Timestamp	R	adrm	o	o	o	-
Timestamp	c	adrm	m	m	m	-
Unsupported	r	arm	c	c	c	-
User-Agent	R	r	m*	m*	-	-
User-Agent	r	r	m*	m*	m*	m*
Vary	r		c	c	-	-
Via	R	amr	o	o	o	-
Via	c	dr	m	m	m	-

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1408
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

WWW-Authenticate	401		m	m	m	-

Table 12: Overview of RTSP header fields (R-W) related to methods GET_PARAMETER, SET_PARAMETER, REDIRECT, and PLAY_NOTIFY.

16.1. Accept

The Accept request-header field can be used to specify certain presentation description and parameter media types [RFC4288] which are acceptable for the response to DESCRIBE and GET_PARAMETER requests.

See Section 20.2.3 for the syntax.

Example of use:
  Accept: application/example ;q=1.0, application/sdp

16.2. Accept-Credentials

The Accept-Credentials header is a request header used to indicate to any trusted intermediary how to handle further secured connections to proxies or servers. See Section 19 for the usage of this header. It MUST NOT be included in server to client requests.

In a request the header MUST contain the method (User, Proxy, or Any) for approving credentials selected by the requester. The method MUST NOT be changed by any proxy, unless it is "proxy" when a proxy MAY change it to "user" to take the role of user approving each further hop. If the method is "User" the header contains zero or more of credentials that the client accepts. The header may contain zero credentials in the first RTSP request to a RTSP server when using the "User" method. This as the client has not yet received any credentials to accept. Each credential MUST consist of one URI identifying the proxy or server, the hash algorithm identifier, and the hash over that agent's DER encoded certificate [RFC5280] in Base64 [RFC4648]. All RTSP clients and proxies MUST implement the SHA-256[FIPS-pub-180-2] algorithm for computation of the hash of the DER encoded certificate. The SHA-256 algorithm is identified by the token "sha-256".

The intention with allowing for other hash algorithms is to enable the future retirement of algorithms that are not implemented somewhere else than here. Thus the definition of future algorithms for this purpose is intended to be extremely limited. A feature tag can be used to ensure that support for the replacement algorithm exist.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1409
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
Example:
Accept-Credentials:User
  "rtsps://proxy2.example.com/";sha-256;exaI19VMbQMOFGClx5rXnPJKVNI=,
  "rtsps://server.example.com/";sha-256;lurbjj5khhB0NhIuOXtt4bBRH1M=
```

16.3. Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but restricts the content-codings, i.e. transformation codings of the message body like gzip compression, that are acceptable in the response.

A server tests whether a content-coding is acceptable, according to an Accept-Encoding field, using these rules:

1. If the content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable, unless it is accompanied by a qvalue of 0. (As defined in section 3.9, a qvalue of 0 means "not acceptable.")

2. The special "*" symbol in an Accept-Encoding field matches any available content-coding not explicitly listed in the header field.

3. If multiple content-codings are acceptable, then the acceptable content-coding with the highest non-zero qvalue is preferred.

4. The "identity" content-coding is always acceptable, i.e. no transformation at all, unless specifically refused because the Accept-Encoding field includes "identity;q=0", or because the field includes "*;q=0" and does not explicitly include the "identity" content-coding. If the Accept-Encoding field-value is empty, then only the "identity" encoding is acceptable.

If an Accept-Encoding field is present in a request, and if the server cannot send a response which is acceptable according to the Accept-Encoding header, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

If no Accept-Encoding field is present in a request, the server MAY assume that the client will accept any content coding. In this case, if "identity" is one of the available content-codings, then the server SHOULD use the "identity" content-coding, unless it has additional information that a different content-coding is meaningful to the client.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1410
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.4. Accept-Language

The Accept-Language request-header field is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request. Note that the language specified applies to the presentation description and any reason phrases, but not the media content.

A language tag identifies a natural language spoken, written, or otherwise conveyed by human beings for communication of information to other human beings. Computer languages are explicitly excluded. The syntax and registry of RTSP 2.0 language tags is the same as that defined by [RFC5646].

Each language-range MAY be given an associated quality value which represents an estimate of the user's preference for the languages specified by that range. The quality value defaults to "q=1". For example:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

would mean: "I prefer Danish, but will accept British English and other types of English." A language-range matches a language-tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first tag character following the prefix is "-". The special range "*", if present in the Accept-Language field, matches every tag not matched by any other range present in the Accept-Language field.

> Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags for which this tag is a prefix. The prefix rule simply allows the use of prefix tags if this is the case.

The language quality factor assigned to a language-tag by the Accept-Language field is the quality value of the longest language-range in the field that matches the language-tag. If no language-range in the field matches the tag, the language quality factor assigned is 0. If no Accept-Language header is present in the request, the server SHOULD assume that all languages are equally acceptable. If an Accept-Language header is present, then all languages which are assigned a quality factor greater than 0 are acceptable.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1411
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.5. Accept-Ranges

The Accept-Ranges general-header field allows indication of the format supported in the Range header. The client MUST include the header in SETUP requests to indicate which formats it support to receive in PLAY and PAUSE responses, and REDIRECT requests. The server MUST include the header in SETUP and 456 error responses to indicate the formats supported for the resource indicated by the request URI. The header MAY be included in GET_PARAMETER request and response pairs. The GET_PARAMETER request MUST contain a Session header to identify the session context the request are related to. The requester and responder will indicate their capabilities regarding Range formats respectively.

```
Accept-Ranges: NPT, SMPTE
```

The syntax is defined in Section 20.2.3.

16.6. Allow

The Allow message-header field lists the methods supported by the resource identified by the Request-URI. The purpose of this field is to strictly inform the recipient of valid methods associated with the resource. An Allow header field MUST be present in a 405 (Method Not Allowed) response. The Allow header MUST also be present in all OPTIONS responses where the content of the header will not include exactly the same methods as listed in the Public header.

The Allow MUST also be included in SETUP and DESCRIBE responses, if the methods allowed for the resource is different than the minimal implementation set.

Example of use:

```
Allow: SETUP, PLAY, SET_PARAMETER, DESCRIBE
```

16.7. Authorization

An RTSP client that wishes to authenticate itself with a server using authentication mechanism from HTTP [RFC2617] , usually, but not necessarily, after receiving a 401 response, does so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

If a request is authenticated and a realm specified, the same credentials SHOULD be valid for all other requests within this realm (assuming that the authentication scheme itself does not require

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1412
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

otherwise, such as credentials that vary according to a challenge value or using synchronized clocks).

When a shared cache (see Section 18) receives a request containing an Authorization field, it MUST NOT return the corresponding response as a reply to any other request, unless one of the following specific exceptions holds:

1. If the response includes the "max-age" cache-control directive, the cache MAY use that response in replying to a subsequent request. But (if the specified maximum age has passed) a proxy cache MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request. (This is the defined behavior for max-age.) If the response includes "max-age=0", the proxy MUST always revalidate it before re-using it.

2. If the response includes the "must-revalidate" cache-control directive, the cache MAY use that response in replying to a subsequent request. But if the response is stale, all caches MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request.

3. If the response includes the "public" cache-control directive, it MAY be returned in reply to any subsequent request.

16.8. Bandwidth

The Bandwidth request-header field describes the estimated bandwidth available to the client, expressed as a positive integer and measured in kilobits per second. The bandwidth available to the client may change during an RTSP session, e.g., due to mobility, congestion, etc.

Clients may not be able to accurately determine the available bandwidth, for example due to that first hop is not a bottleneck. For example most local area networks (LAN) will not be a bottleneck if the server is not in the same LAN. Thus link speeds of WLAN or Ethernet networks are normally not a basis for estimating the available bandwidth. Cellular devices or other devices directly connected to an modem or connection enabling device may more accurately estimate the bottleneck bandwidth and what is reasonable share of it for RTSP controlled media. The client will also need to take into account other traffic sharing the bottleneck. For example by only assigning a certain fraction to RTSP and its media streams. It is RECOMMENDED that only clients that has accurate and explicit information about bandwidth bottlenecks uses this header.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1413
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

This header is not a substitute for proper congestion control. Only a method providing an initial estimate and coarsely determine if the selected content can be delivered at all.

Example:
  Bandwidth: 62360

16.9. Blocksize

The Blocksize request-header field is sent from the client to the media server asking the server for a particular media packet size. This packet size does not include lower-layer headers such as IP, UDP, or RTP. The server is free to use a blocksize which is lower than the one requested. The server MAY truncate this packet size to the closest multiple of the minimum, media-specific block size, or override it with the media-specific size if necessary. The block size MUST be a positive decimal number, measured in octets. The server only returns an error (4xx) if the value is syntactically invalid.

16.10. Cache-Control

The Cache-Control general-header field is used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain.

Cache directives MUST be passed through by a proxy or gateway application, regardless of their significance to that application, since the directives may be applicable to all recipients along the request/response chain. It is not possible to specify a cache-directive for a specific cache.

Cache-Control should only be specified in a DESCRIBE, GET_PARAMETER, SET_PARAMETER and SETUP request and its response. Note: Cache-Control does not govern only the caching of responses as for HTTP, instead it also applies to the media stream identified by the SETUP request. The RTSP requests are generally not cacheable, for further information see Section 18. Below is the description of the cache directives that can be included in the Cache-Control header.

no-cache: Indicates that the media stream MUST NOT be cached anywhere. This allows an origin server to prevent caching even by caches that have been configured to return stale responses to client requests. Note, there is no security function enforcing that the content can't be cached.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1414
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

public: Indicates that the media stream is cacheable by any cache.

private: Indicates that the media stream is intended for a single user and MUST NOT be cached by a shared cache. A private (non-shared) cache may cache the media streams.

no-transform: An intermediate cache (proxy) may find it useful to convert the media type of a certain stream. A proxy might, for example, convert between video formats to save cache space or to reduce the amount of traffic on a slow link. Serious operational problems may occur, however, when these transformations have been applied to streams intended for certain kinds of applications. For example, applications for medical imaging, scientific data analysis and those using end-to-end authentication all depend on receiving a stream that is bit-for-bit identical to the original media stream. Therefore, if a response includes the no-transform directive, an intermediate cache or proxy MUST NOT change the encoding of the stream. Unlike HTTP, RTSP does not provide for partial transformation at this point, e.g., allowing translation into a different language.

only-if-cached: In some cases, such as times of extremely poor network connectivity, a client may want a cache to return only those media streams that it currently has stored, and not to receive these from the origin server. To do this, the client may include the only-if-cached directive in a request. If it receives this directive, a cache SHOULD either respond using a cached media stream that is consistent with the other constraints of the request, or respond with a 504 (Gateway Timeout) status. However, if a group of caches is being operated as a unified system with good internal connectivity, such a request MAY be forwarded within that group of caches.

max-stale: Indicates that the client is willing to accept a media stream that has exceeded its expiration time. If max-stale is assigned a value, then the client is willing to accept a response that has exceeded its expiration time by no more than the specified number of seconds. If no value is assigned to max-stale, then the client is willing to accept a stale response of any age.

min-fresh: Indicates that the client is willing to accept a media stream whose freshness lifetime is no less than its current age plus the specified time in seconds. That is, the client wants a response that will still be fresh for at least the specified number of seconds.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1415
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

must-revalidate: When the must-revalidate directive is present in a SETUP response received by a cache, that cache MUST NOT use the entry after it becomes stale to respond to a subsequent request without first revalidating it with the origin server. That is, the cache is required to do an end-to-end revalidation every time, if, based solely on the origin server's Expires, the cached response is stale.)

proxy-revalidate: The proxy-revalidate directive has the same meaning as the must-revalidate directive, except that it does not apply to non-shared user agent caches. It can be used on a response to an authenticated request to permit the user's cache to store and later return the response without needing to revalidate it (since it has already been authenticated once by that user), while still requiring proxies that service many users to revalidate each time (in order to make sure that each user has been authenticated). Note that such authenticated responses also need the public cache control directive in order to allow them to be cached at all.

max-age: When an intermediate cache is forced, by means of a max-age=0 directive, to revalidate its own cache entry, and the client has supplied its own validator in the request, the supplied validator might differ from the validator currently stored with the cache entry. In this case, the cache MAY use either validator in making its own request without affecting semantic transparency.

However, the choice of validator might affect performance. The best approach is for the intermediate cache to use its own validator when making its request. If the server replies with 304 (Not Modified), then the cache can return its now validated copy to the client with a 200 (OK) response. If the server replies with a new message body and cache validator, however, the intermediate cache can compare the returned validator with the one provided in the client's request, using the strong comparison function. If the client's validator is equal to the origin server's, then the intermediate cache simply returns 304 (Not Modified). Otherwise, it returns the new message body with a 200 (OK) response.

16.11. Connection

The Connection general-header field allows the sender to specify options that are desired for that particular connection and MUST NOT be communicated by proxies over further connections.

RTSP 2.0 proxies MUST parse the Connection header field before a message is forwarded and, for each connection-token in this field,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1416
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

remove any header field(s) from the message with the same name as the connection-token. Connection options are signaled by the presence of a connection-token in the Connection header field, not by any corresponding additional header field(s), since the additional header field may not be sent if there are no parameters associated with that connection option.

Message headers listed in the Connection header MUST NOT include end-to-end headers, such as Cache-Control.

RTSP 2.0 defines the "close" connection option for the sender to signal that the connection will be closed after completion of the response. For example, Connection: close in either the request or the response header fields indicates that the connection SHOULD NOT be considered 'persistent' (Section 10.2) after the current request/response is complete.

The use of the connection option "close" in RTSP messages SHOULD be limited to error messages when the server is unable to recover and therefore see it necessary to close the connection. The reason is that the client has the choice of continuing using a connection indefinitely, as long as it sends valid messages.

16.12. Connection-Credentials

The Connection-Credentials response header is used to carry the chain of credentials of any next hop that need to be approved by the requester. It MUST only be used in server to client responses.

The Connection-Credentials header in an RTSP response MUST, if included, contain the credential information (in form of a list of certificates providing the chain of certification) of the next hop that an intermediary needs to securely connect to. The header MUST include the URI of the next hop (proxy or server) and a base64 [RFC4648] encoded binary structure containing a sequence of DER encoded X.509v3 certificates[RFC5280] .

The binary structure starts with the number of certificates (NR_CERTS) included as a 16 bit unsigned integer. This is followed by NR_CERTS number of 16 bit unsigned integers providing the size in octets of each DER encoded certificate. This is followed by NR_CERTS number of DER encoded X.509v3 certificates in a sequence (chain). The proxy or server's certificate must come first in the structure. Each following certificate must directly certify the one preceding it. Because certificate validation requires that root keys be distributed independently, the self-signed certificate which specifies the root certificate authority may optionally be omitted from the chain, under the assumption that the remote end must already

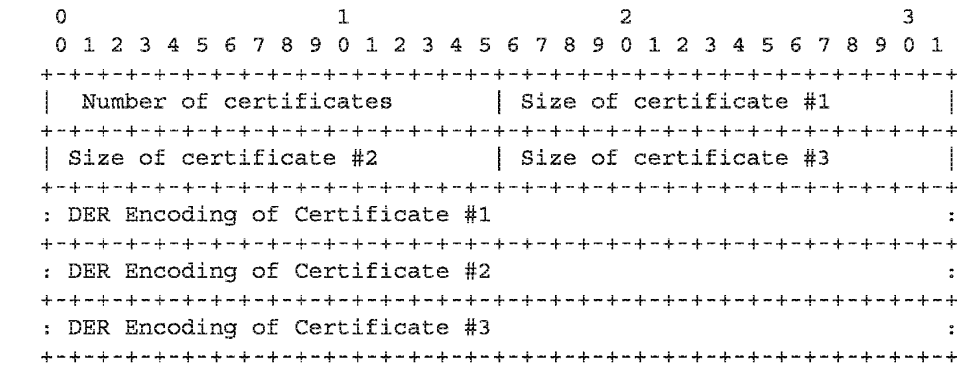
Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1417
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

possess it in order to validate it in any case.

Example:

Connection-Credentials:"rtsps://proxy2.example.com/";MIIDNTCC...

Where MIIDNTCC... is a BASE64 encoding of the following structure:



16.13. Content-Base

The Content-Base message-header field may be used to specify the base URI for resolving relative URIs within the message body.

Content-Base: rtsp://media.example.com/movie/twister/

If no Content-Base field is present, the base URI of an message body is defined either by its Content-Location (if that Content-Location URI is an absolute URI) or the URI used to initiate the request, in that order of precedence. Note, however, that the base URI of the contents within the message-body may be redefined within that message-body.

16.14. Content-Encoding

The Content-Encoding header field is used as a modifier to the media-type. When present, its value indicates what additional content codings have been applied to the message body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

The content-coding is a characteristic of the message body identified by the Request-URI. Typically, the message body is stored with this

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1418
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

encoding and is only decoded before rendering or analogous usage. However, a non-transparent proxy MAY modify the content-coding if the new coding is known to be acceptable to the recipient, unless the "no-transform" cache-control directive is present in the message.

If the content-coding of an message body is not "identity", then the response MUST include a Content-Encoding Message-body header that lists the non-identity content-coding(s) used.

If the content-coding of an message body in a request message is not acceptable to the origin server, the server SHOULD respond with a status code of 415 (Unsupported Media Type).

If multiple encodings have been applied to a message body, the content codings MUST be listed in the order in which they were applied, first to last from left to right. Additional information about the encoding parameters MAY be provided by other header fields not defined by this specification.

16.15. Content-Language

The Content-Language header field describes the natural language(s) of the intended audience for the enclosed message body. Note that this might not be equivalent to all the languages used within the message body.

Language tags are mentioned in Section 16.4. The primary purpose of Content-Language is to allow a user to identify and differentiate entities according to the user's own preferred language. Thus, if the body content is intended only for a Danish-literate audience, the appropriate field is

```
Content-Language: da
```

If no Content-Language is specified, the default is that the content is intended for all language audiences. This might mean that the sender does not consider it to be specific to any natural language, or that the sender does not know for which language it is intended.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi," presented simultaneously in the original Maori and English versions, would call for

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1419
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
Content-Language: mi, en
```

However, just because multiple languages are present within an message body does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin," which is clearly intended to be used by an English-literate audience. In this case, the Content-Language would properly only include "en".

Content-Language MAY be applied to any media type -- it is not limited to textual documents.

16.16. Content-Length

The Content-Length general-header field contains the length of the message body of the RTSP message (i.e. after the double CRLF following the last header). Unlike HTTP, it MUST be included in all messages that carry a message body beyond the header portion of the RTSP message. If it is missing, a default value of zero is assumed. Any Content-Length greater than or equal to zero is a valid value.

16.17. Content-Location

The Content-Location header field MAY be used to supply the resource location for the message body enclosed in the message when that body is accessible from a location separate from the requested resource's URI. A server SHOULD provide a Content-Location for the variant corresponding to the response message body; especially in the case where a resource has multiple variants associated with it, and those entities actually have separate locations by which they might be individually accessed, the server SHOULD provide a Content-Location for the particular variant which is returned.

The Content-Location value is not a replacement for the original requested URI; it is only a statement of the location of the resource corresponding to this particular variant at the time of the request. Future requests MAY specify the Content-Location URI as the request URI if the desire is to identify the source of that particular variant. This is useful if the RTSP agent desires verify if the resource variant is current through a conditional request.

A cache cannot assume that an message body with a Content-Location different from the URI used to retrieve it can be used to respond to later requests on that Content-Location URI. However, the Content-Location can be used to differentiate between multiple variants retrieved from a single requested resource.

If the Content-Location is a relative URI, the relative URI is

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1420
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

interpreted relative to the Request-URI.

Note, that Content-Location can be used in some cases to derive the base-URI for relative URI present in session description formats. This needs to be taken into account when Content-Location is used. The easiest way to avoid needing to consider that issue is to include the Content-Base whenever the Content-Location is included.

Note also, when using Media Tags in conjunction with Content-Location it is important that the different versions have different MTags, even if provided under different Content-Location URIs. This as they have still been provided under the same request URI.

Note also, as in most cases as the URI used in the DESCRIBE and the SETUP requests are different the URI provided in a DESCRIBE Content-Location response can't directly be used in a SETUP request. Instead the extra step of resolving URIs combined with the media descriptions indication, like with SDP's a=control attribute.

16.18. Content-Type

The Content-Type header indicates the media type of the message body sent to the recipient. Note that the content types suitable for RTSP are likely to be restricted in practice to presentation descriptions and parameter-value types.

16.19. CSeq

The CSeq general-header field specifies the sequence number for an RTSP request-response pair. This field MUST be present in all requests and responses. For every RTSP request containing the given sequence number, the corresponding response will have the same number. Any retransmitted request MUST contain the same sequence number as the original (i.e. the sequence number is not incremented for retransmissions of the same request). For each new RTSP request the CSeq value MUST be incremented by one. The initial sequence number MAY be any number, however, it is RECOMMENDED to start at 0. Each sequence number series is unique between each requester and responder, i.e. the client has one series for its request to a server and the server has another when sending request to the client. Each requester and responder is identified with its network address.

Proxies that aggregate several sessions on the same transport will have to ensure that the requests sent towards a particular server have a joint sequence number space, i.e., they will regularly need to renumber the CSeq header field in requests (from proxy to server) and responses (from server to proxy) to fulfill the rules for the header. The proxy MUST increase the CSeq by one for each request it

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1421
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

transmits, without regard of different sessions.

Example:
CSeq: 239

16.20. Date

The Date header field represents the date and time at which the message was originated. The inclusion of the Date header in RTSP message follows these rules:

- An RTSP message, sent either by the client or the server, containing a body MUST include a Date header, if the sending host has a clock;

- Clients and servers are RECOMMENDED to include a Date header in all other RTSP messages, if the sending host has a clock;

- If the server does not have a clock that can provide a reasonable approximation of the current time, its responses MUST NOT include a Date header field. In this case, this rule MUST be followed: Some origin server implementations might not have a clock available. An origin server without a clock MUST NOT assign Expires or Last- Modified values to a response, unless these values were associated with the resource by a system or user with a reliable clock. It MAY assign an Expires value that is known, at or before server configuration time, to be in the past (this allows "pre-expiration" of responses without storing separate Expires values for each resource).

A received message that does not have a Date header field MUST be assigned one by the recipient if the message will be cached by that recipient . An RTSP implementation without a clock MUST NOT cache responses without revalidating them on every use. An RTSP cache, especially a shared cache, SHOULD use a mechanism, such as NTP, to synchronize its clock with a reliable external standard.

The RTSP-date sent in a Date header SHOULD NOT represent a date and time subsequent to the generation of the message. It SHOULD represent the best available approximation of the date and time of message generation, unless the implementation has no means of generating a reasonably accurate date and time. In theory, the date ought to represent the moment just before the message body is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1422
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.21. Expires

The Expires message-header field gives a date and time after which the description or media-stream should be considered stale. The interpretation depends on the method:

DESCRIBE response: The Expires header indicates a date and time after which the presentation description (body) SHOULD be considered stale.

SETUP response: The Expires header indicate a date and time after which the media stream SHOULD be considered stale.

A stale cache entry may not normally be returned by a cache (either a proxy cache or an user agent cache) unless it is first validated with the origin server (or with an intermediate cache that has a fresh copy of the message body). See Section 18 for further discussion of the expiration model.

The presence of an Expires field does not imply that the original resource will change or cease to exist at, before, or after that time.

The format is an absolute date and time as defined by RTSP-date. An example of its use is

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

RTSP/2.0 clients and caches MUST treat other invalid date formats, especially including the value "0", as having occurred in the past (i.e., already expired).

To mark a response as "already expired," an origin server should use an Expires date that is equal to the Date header value. To mark a response as "never expires," an origin server SHOULD use an Expires date approximately one year from the time the response is sent. RTSP/2.0 servers SHOULD NOT send Expires dates more than one year in the future.

16.22. From

The From request-header field, if given, SHOULD contain an Internet e-mail address for the human user who controls the requesting user agent. The address SHOULD be machine-usable, as defined by "mailbox" in [RFC1123].

This header field MAY be used for logging purposes and as a means for identifying the source of invalid or unwanted requests. It SHOULD NOT be used as an insecure form of access protection. The

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1423
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsibility for the method performed. In particular, robot agents SHOULD include this header so that the person responsible for running the robot can be contacted if problems occur on the receiving end.

The Internet e-mail address in this field MAY be separate from the Internet host which issued the request. For example, when a request is passed through a proxy the original issuer's address SHOULD be used.

The client SHOULD NOT send the From header field without the user's approval, as it might conflict with the user's privacy interests or their site's security policy. It is strongly recommended that the user be able to disable, enable, and modify the value of this field at any time prior to a request.

16.23. If-Match

The If-Match request-header field is especially useful for ensuring the integrity of the presentation description, independent of how the presentation description was received. The presentation description can be fetched via means external to RTSP (such as HTTP) or via the DESCRIBE message. In the case of retrieving the presentation description via RTSP, the server implementation is guaranteeing the integrity of the description between the time of the DESCRIBE message and the SETUP message. By including the MTag given in or with the session description in a If-Match header part of the SETUP request, the client ensures that resources set up are matching the description. A SETUP request with the If-Match header for which the MTag validation check fails, MUST response using 412 (Precondition Failed).

This validation check is also very useful if a session has been redirected from one server to another.

16.24. If-Modified-Since

The If-Modified-Since request-header field is used with the DESCRIBE and SETUP methods to make them conditional. If the requested variant has not been modified since the time specified in this field, a description will not be returned from the server (DESCRIBE) or a stream will not be set up (SETUP). Instead, a 304 (Not Modified) response MUST be returned without any message-body.

An example of the field is:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1424
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.25. If-None-Match

This request header can be used with one or several message body tags to make DESCRIBE requests conditional. A client that has one or more message bodies previously obtained from the resource, can verify that none of those entities is current by including a list of their associated message body tags in the If-None-Match header field. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. As a special case, the value "*" matches any current entity of the resource.

if any of the message body tags match the message body tag of the message body that would have been returned in the response to a similar DESCRIBE request (without the If-None-Match header) on that resource, or if "*" is given and any current entity exists for that resource, then the server MUST NOT perform the requested method, unless required to do so because the resource's modification date fails to match that supplied in an If-Modified-Since header field in the request. Instead, if the request method was DESCRIBE, the server SHOULD respond with a 304 (Not Modified) response, including the cache-related header fields (particularly MTag) of one of the message bodies that matched. For all other request methods, the server MUST respond with a status of 412 (Precondition Failed).

See Section 18.1.3 for rules on how to determine if two message body tags match.

If none of the message body tags match, then the server MAY perform the requested method as if the If-None-Match header field did not exist, but MUST also ignore any If-Modified-Since header field(s) in the request. That is, if no message body tags match, then the server MUST NOT return a 304 (Not Modified) response.

If the request would, without the If-None-Match header field, result in anything other than a 2xx or 304 status, then the If-None-Match header MUST be ignored. (See Section 18.1.4 for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)

The result of a request having both an If-None-Match header field and an If-Match header field is unspecified and MUST be considered an illegal request.

16.26. Last-Modified

The Last-Modified message-header field indicates the date and time at which the origin server believes the presentation description or

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1425
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

media stream was last modified. For the method DESCRIBE, the header field indicates the last modification date and time of the description, for SETUP that of the media stream.

An origin server MUST NOT send a Last-Modified date which is later than the server's time of message origination. In such cases, where the resource's last modification would indicate some time in the future, the server MUST replace that date with the message origination date.

An origin server SHOULD obtain the Last-Modified value of the message body as close as possible to the time that it generates the Date value of its response. This allows a recipient to make an accurate assessment of the message body's modification time, especially if the message body changes near the time that the response is generated.

RTSP servers SHOULD send Last-Modified whenever feasible.

16.27. Location

The Location response-header field is used to redirect the recipient to a location other than the Request-URI for completion of the request or identification of a new resource. For 3xx responses, the location SHOULD indicate the server's preferred URI for automatic redirection to the resource. The field value consists of a single absolute URI.

Note: The Content-Location header field (Section 16.17) differs from Location in that the Content-Location identifies the original location of the message body enclosed in the request. It is therefore possible for a response to contain header fields for both Location and Content-Location. Also, see Section 18.2 for cache requirements of some methods.

16.28. Media-Properties

This general header is used in SETUP response or PLAY_NOTIFY requests to indicate the media's properties that currently are applicable to the RTSP session. PLAY_NOTIFY MAY be used to modify these properties at any point. However, the client SHOULD have received the update prior to that any action related to the new media properties take effect. For aggregated sessions, the Media-Properties header will be returned in each SETUP response. The header received in the latest response is the one that applies on the whole session from this point until any future update. The header MAY be included without value in GET_PARAMETER requests to the server with a Session header included to query the current Media-Properties for the session. The responder MUST include the current session's media properties.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1426
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

The media properties expressed by this header is the one applicable to all media in the RTSP session. For aggregated sessions, the header expressed the combined media-properties. As a result aggregation of media MAY result in a change of the media properties, and thus the content of the Media-Properties header contained in subsequent SETUP responses.

The header contains a list of property values that are applicable to the currently setup media or aggregate of media as indicated by the RTSP URI in the request. No ordering are enforced within the header. Property values should be grouped into a single group that handles a particular orthogonal property. Values or groups that express multiple properties SHOULD NOT be used. The list of properties that can be expressed MAY be extended at any time. Unknown property values MUST be ignored.

This specification defines the following 4 groups and their property values:

Random Access:

- Random-Access: Indicates that random access is possible. May optionally include a floating point value in seconds indicating the longest duration between any two random access points in the media.

- Begining-Only: Seeking is limited to the beginning only.

- No-Seeking: No seeking is possible.

Content Modifications:

- Immutable: The content will not be changed during the life-time of the RTSP session.

- Dynamic: The content may be changed based on external methods or triggers

- Time-Progressing The media accessible progress as wallclock time progresses.

Retention:

- Unlimited: Content will be retained for the duration of the life-time of the RTSP session.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1427
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Time-Limited: Content will be retained at least until the specified wallclock time. The time must be provided in the absolute time format specified in Section 4.6.

Time-Duration Each individual media unit is retained for at least the specified time duration. This definition allows for retaining data with a time based sliding window. The time duration is expressed as floating point number in seconds. 0.0 is a valid value as this indicates that no data is retained in a time-progressing session.

Supported Scale:

Scales: A quoted comma separated list of one or more decimal values or ranges of scale values supported by the content in arbitrary order. A range has a start and stop value separated by a colon. A range indicates that the content supports fine grained selection of scale values. Fine grained allows for steps at least as small as one tenth of a scale value. Negative values are supported. The value 0 have no meaning and must not be used.

Examples of this header for on-demand content and a live stream without recording are:

```
On-demand:
Media-Properties: Random-Access=2.5s, Unlimited, Immutable,
     Scales="-20, -10, -4, 0.5:1.5, 4, 8, 10, 15, 20"

Live stream without recording/timeshifting:
Media-Properties: No-Seeking, Time-Progressing, Time-Duration=0.0
```

16.29. Media-Range

The Media-Range general header is used to give the range of the media at the time of sending the RTSP message. This header MUST be included in SETUP response, and PLAY and PAUSE response for media that are Time-Progressing, and PLAY and PAUSE response after any change for media that are Dynamic, and in PLAY_NOTIFY request that are sent due to Media-Property-Update. Media-Range header without any range specifications MAY be included in GET_PARAMETER requests to the server to request the current range. The server MUST in this case include the current range at the time of sending the response.

The header MUST include range specifications for all time formats supported for the media, as indicated in Accept-Ranges header (Section 16.5) when setting up the media. The server MAY include more than one range specification of any given time format to

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1428
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

indicate media that has non-continuous range.

For media that has the Time-Progressing property, the Media-Range values will only be valid for the particular point in time when it was issued. As wallclock progresses so will also the media range. However, it shall be assumed that media time progress in direct relationship to wallclock time (with the exception of clock skew) so that a reasonably accurate estimation of the media range can be calculated.

16.30. MTag

The MTag response header MAY be included in DESCRIBE, GET_PARAMETER or SETUP responses. The message body tags (Section 4.8) returned in a DESCRIBE response, and the one in SETUP refers to the presentation, i.e. both the returned session description and the media stream. This allows for verification that one has the right session description to a media resource at the time of the SETUP request. However, it has the disadvantage that a change in any of the parts results in invalidation of all the parts.

If the MTag is provided both inside the message body, e.g. within the "a=mtag" attribute in SDP, and in the response message, then both tags MUST be identical. It is RECOMMENDED that the MTag is primarily given in the RTSP response message, to ensure that caches can use the MTag without requiring content inspection. However, for session descriptions that are distributed outside of RTSP, for example using HTTP, etc. it will be necessary to include the message body tag in the session description as specified in Appendix D.1.9.

SETUP and DESCRIBE requests can be made conditional upon the MTag using the headers If-Match (Section 16.23) and If-None-Match (Section 16.25).

16.31. Notify-Reason

The Notify Reason header is solely used in the PLAY_NOTIFY method. It indicates the reason why the server has sent the asynchronous PLAY_NOTIFY request (see Section 13.5).

16.32. Pipelined-Requests

The Pipelined-Requests general header is used to indicate that a request is to be executed in the context created by a previous request(s). The primary usage of this header is to allow pipelining of SETUP requests so that any additional SETUP request after the first one does not need to wait for the session ID to be sent back to the requesting agent. The header contains a unique identifier that

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1429
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

is scoped by the persistent connection used to send the requests.

Upon receiving a request with the Pipelined-Requests the responding agent MUST look up if there exists a binding between this Pipelined-Requests identifier for the current persistent connection and an RTSP session ID. If that exists then the received request is processed the same way as if it contained the Session header with the found session ID. If there does not exist a mapping and no Session header is included in the request, the responding agent MUST create a binding upon the successful completion of a session creating request, i.e. SETUP. A binding MUST NOT be created, if the request failed to create an RTSP session. In case the request contains both a Session header and the Pipelined-Requests header the Pipelined-Requests MUST be ignored.

Note: Based on the above definition at least the first request containing a new unique Pipelined-Requests will be required to be a SETUP request (unless the protocol is extended with new methods of creating a session). After that first one, additional SETUP requests or request of any type using the RTSP session context may include the Pipelined-Requests header.

When responding to any request that contained the Pipelined-Requests header the server MUST also include the Session header when a binding to a session context exist. A RTSP agent that knows the session ID SHOULD NOT use the Pipelined-Requests header in any request and only use the Session header. This as the Session identifier is persistent across transport contexts, like TCP connections, which the Pipelined-Requests identifier is not.

The RTSP agent sending the request with a Pipelined-Requests header has the responsibility for using a unique and previously unused identifier within the transport context. Currently only a TCP connection is defined as such transport context. A server MUST delete the Pipelined-Requests identifier and its binding to a session upon the termination of that session. Despite the previous mandate, RTSP agents are RECOMMENDED to not reuse identifiers to allow for better error handling and logging.

RTSP Proxies may need to translate Pipelined-Requests identifier values from incoming request to outgoing to allow for aggregation of requests onto a persistent connection.

16.33. Proxy-Authenticate

The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1430
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

parameters applicable to the proxy for this Request-URI.

The HTTP access authentication process is described in [RFC2617]. Unlike WWW-Authenticate, the Proxy-Authenticate header field applies only to the current connection and SHOULD NOT be passed on to downstream agents. However, an intermediate proxy might need to obtain its own credentials by requesting them from the downstream agent, which in some circumstances will appear as if the proxy is forwarding the Proxy-Authenticate header field.

16.34. Proxy-Authorization

The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy which requires authentication. The Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

The HTTP access authentication process is described in [RFC2617]. Unlike Authorization, the Proxy-Authorization header field applies only to the next outbound proxy that demanded authentication using the Proxy- Authenticate field. When multiple proxies are used in a chain, the Proxy-Authorization header field is consumed by the first outbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

16.35. Proxy-Require

The Proxy-Require request-header field is used to indicate proxy-sensitive features that MUST be supported by the proxy. Any Proxy-Require header features that are not supported by the proxy MUST be negatively acknowledged by the proxy to the client using the Unsupported header. The proxy MUST use the 551 (Option Not Supported) status code in the response. Any feature-tag included in the Proxy-Require does not apply to the end-point (server or client). To ensure that a feature is supported by both proxies and servers the tag needs to be included in also a Require header.

See Section 16.41 for more details on the mechanics of this message and a usage example. See discussion in the proxies section (Section 17.1) about when to consider that a feature requires proxy support.

Example of use:

```
Proxy-Require: play.basic
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1431
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.36. Proxy-Supported

The Proxy-Supported header field enumerates all the extensions supported by the proxy using feature-tags. The header carries the intersection of extensions supported by the forwarding proxies. The Proxy-Supported header MAY be included in any request by a proxy. It MUST be added by any proxy if the Supported header is present in a request. When present in a request, the receiver MUST in the response copy the received Proxy-Supported header.

The Proxy-Supported header field contains a list of feature-tags applicable to proxies, as described in Section 4.7. The list are the intersection of all feature-tags understood by the proxies. To achieve an intersection, the proxy adding the Proxy-Supported header includes all proxy feature-tags it understands. Any proxy receiving a request with the header, checks the list and removes any feature-tag it do not support. A Proxy-Supported header present in the response MUST NOT be touched by the proxies.

Example:

```
  C->P1: OPTIONS rtsp://example.com/ RTSP/2.0
         Supported: foo, bar, blech
         User-Agent: PhonyClient/1.2

 P1->P2: OPTIONS rtsp://example.com/ RTSP/2.0
         Supported: foo, bar, blech
         Proxy-Supported: proxy-foo, proxy-bar, proxy-blech
         Via: 2.0 pro.example.com

 P2->S:  OPTIONS rtsp://example.com/ RTSP/2.0
         Supported: foo, bar, blech
         Proxy-Supported: proxy-foo, proxy-blech
         Via: 2.0 pro.example.com, 2.0 prox2.example.com

  S->C:  RTSP/2.0 200 OK
         Supported: foo, bar, baz
         Proxy-Supported: proxy-foo, proxy-blech
         Public: OPTIONS, SETUP, PLAY, PAUSE, TEARDOWN
         Via: 2.0 pro.example.com, 2.0 prox2.example.com
```

16.37. Public

The Public response header field lists the set of methods supported by the response sender. This header applies to the general capabilities of the sender and its only purpose is to indicate the sender's capabilities to the recipient. The methods listed may or may not be applicable to the Request-URI; the Allow header field (Section 16.6) MAY be used to indicate methods allowed for a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1432
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

particular URI.

Example of use:
   Public: OPTIONS, SETUP, PLAY, PAUSE, TEARDOWN

In the event that there are proxies between the sender and the recipient of a response, each intervening proxy MUST modify the Public header field to remove any methods that are not supported via that proxy. The resulting Public header field will contain an intersection of the sender's methods and the methods allowed through by the intervening proxies.

> In general, proxies should allow all methods to transparently pass through from the sending RTSP agent to the receiving RTSP agent, but there may be cases where this is not desirable for a given proxy. Modification of the Public response header field by the intervening proxies ensures that the request sender gets an accurate response indicating the methods that can be used on the target agent via the proxy chain.

16.38. Range

The Range header specifies a time range in PLAY (Section 13.4), PAUSE (Section 13.6), SETUP (Section 13.3), REDIRECT (Section 13.10), and PLAY_NOTIFY (Section 13.5) requests and responses. It MAY be included in GET_PARAMETER requests from the client to the server with only a Range format and no value to request the current media position, whether the session is in Play or Ready state in the included format. The server SHALL, if supporting the range format, respond with the current playing point or pause point as the start of the range. If an explicit stop point was used in the previous PLAY request, then that value shall be included as stop point. Note that if the server is currently under any type of media playback manipulation affecting the interpretation of Range, like Scale, that is also required to be included in any GET_PARAMETER response to provide complete information.

The range can be specified in a number of units. This specification defines smpte (Section 4.4), npt (Section 4.5), and clock (Section 4.6) range units. While byte ranges [H14.35.1] and other extended units MAY be used, their behavior is unspecified since they are not normally meaningful in RTSP. Servers supporting the Range header MUST understand the NPT range format and SHOULD understand the SMPTE range format. If the Range header is sent in a time format that is not understood, the recipient SHOULD return 456 (Header Field Not Valid for Resource) and include an Accept-Ranges header indicating the supported time formats for the given resource.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1433
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Example:
  Range: clock=19960213T143205Z-

The Range header contains a range of one single range format. A range is a half-open interval with a start and an end point, including the start point, but excluding the end point. A range may either be fully specified with explicit values for start point and end point, or have either start or end point be implicit. An implicit start point indicates the session's pause point, and if no pause point is set the start of the content. An implicit end point indicates the end of the content. The usage of both implicit start and end point is not allowed in the same range header, however, the exclusion of the range header has that meaning, i.e. from pause point (or start) until end of content.

Regarding the half-open intervals; a range of A-B starts exactly at time A, but ends just before B. Only the start time of a media unit such as a video or audio frame is relevant. For example, assume that video frames are generated every 40 ms. A range of 10.0-10.1 would include a video frame starting at 10.0 or later time and would include a video frame starting at 10.08, even though it lasted beyond the interval. A range of 10.0-10.08, on the other hand, would exclude the frame at 10.08.

Please note the difference between NPT time scales' "now" and an implicit start value. Implicit value reference the current pause-point. While "now" is the currently ongoing time. In a time-progressing session with recording (retention for some or full time) the pause point may be 2 min into the session while now could be 1 hour into the session.

By default, range intervals increase, where the second point is larger than the first point.

Example:
    Range: npt=10-15

However, range intervals can also decrease if the Scale header (see Section 16.44) indicates a negative scale value. For example, this would be the case when a playback in reverse is desired.

Example:
    Scale: -1
    Range: npt=15-10

Decreasing ranges are still half open intervals as described above. Thus, for range A-B, A is closed and B is open. In the above example, 15 is closed and 10 is open. An exception to this rule is

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1434
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

the case when B=0 in a decreasing range. In this case, the range is closed on both ends, as otherwise there would be no way to reach 0 on a reverse playback for formats that have such a notion, like NPT and SMPTE.

Example:
```
Scale: -1
Range: npt=15-0
```

In this range both 15 and 0 are closed.

A decreasing range interval without a corresponding negative Scale header is not valid.

16.39. Referrer

The Referrer request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained. The URI refers to that of the presentation description, typically retrieved via HTTP. The Referrer request-header allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referrer field MUST NOT be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

If the field value is a relative URI, it SHOULD be interpreted relative to the Request-URI. The URI MUST NOT include a fragment.

Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referrer field is sent. For example, a streaming client could have a toggle switch for openly/anonymously, which would respectively enable/disable the sending of Referee and From information.

Clients SHOULD NOT include a Referee header field in a (non-secure) RTSP request if the referring page was transferred with a secure protocol.

16.40. Request-Status

This request header is used to indicate the end result for requests that takes time to complete, such a PLAY (Section 13.4). It is sent in PLAY_NOTIFY (Section 13.5) with the end-of-stream reason to report how the PLAY request concluded, either in success or in failure. The header carries a reference to the request it reports on using the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1435
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

CSeq number for the session indicated by the Session header in the request. It provides both a numerical status code (according to Section 8.1.1) and a human readable reason phrase.

Example:
Request-Status: cseq=63 status=500 reason="Media data unavailable"

16.41. Require

The Require request-header field is used by clients or servers to ensure that the other end-point supports features that are required in respect to this request. It can also be used to query if the other end-point supports certain features, however, the use of the Supported (Section 16.49) is much more effective in this purpose. The server MUST respond to this header by using the Unsupported header to negatively acknowledge those feature-tags which are NOT supported. The response MUST use the error code 551 (Option Not Supported). This header does not apply to proxies, for the same functionality in respect to proxies see Proxy-Require header (Section 16.35) with the exception of media modifying proxies. Media modifying proxies due to their nature of handling media in a way that is very similar to what a server, do need to understand also the server features to correctly serve the client.

> This is to make sure that the client-server interaction will proceed without delay when all features are understood by both sides, and only slow down if features are not understood (as in the example below). For a well-matched client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms. In addition, it also removes state ambiguity when the client requires features that the server does not understand.

Example (Not complete):

```
C->S:   SETUP rtsp://server.com/foo/bar/baz.rm RTSP/2.0
        CSeq: 302
        Require: funky-feature
        Funky-Parameter: funkystuff

S->C:   RTSP/2.0 551 Option not supported
        CSeq: 302
        Unsupported: funky-feature
```

In this example, "funky-feature" is the feature-tag which indicates to the client that the fictional Funky-Parameter field is required. The relationship between "funky-feature" and Funky-Parameter is not communicated via the RTSP exchange, since that relationship is an immutable property of "funky-feature" and thus should not be

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1436
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

transmitted with every exchange.

Proxies and other intermediary devices MUST ignore this header. If a particular extension requires that intermediate devices support it, the extension should be tagged in the Proxy-Require field instead (see Section 16.35). See discussion in the proxies section (Section 17.1) about when to consider that a feature requires proxy support.

16.42. Retry-After

The Retry-After response-header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. This field MAY also be used with any 3xx (Redirection) response to indicate the minimum time the user-agent is asked wait before issuing the redirected request. The value of this field can be either an RTSP-date or an integer number of seconds (in decimal) after the time of the response.

Example:
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120

In the latter example, the delay is 2 minutes.

16.43. RTP-Info

The RTP-Info general field is used to set RTP-specific parameters in the PLAY and GET_PARAMETER responses or a PLAY_NOTIFY and GET_PARAMETER requests. For streams using RTP as transport protocol the RTP-Info header SHOULD be part of a 200 response to PLAY.

> The exclusion of the RTP-Info in a PLAY response for RTP transported media will result in that a client needs to synchronize the media streams using RTCP. This may have negative impact as the RTCP can be lost, and does not need to be particularly timely in their arrival. Also functionality as informing the client from which packet a seek has occurred is affected.

The RTP-Info MAY be included in SETUP responses to provide synchronization information when changing transport parameters, see Section 13.3. The RTP-Info header and the Range header MAY be included in a GET_PARAMETER request from client to server without any values to request the current playback point and corresponding. RTP synchronization information. When the RTP-Info header is included in a Request also the Range header MUST be included (Note, Range header only MAY be used). The server response SHALL include both the Range

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1437
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

header and the RTP-Info header. If the session is in Play state, then the value of the Range header SHALL be filled in with the current playback point and with the corresponding RTP-Info values. If the server is another state, no values are included in the RTP-Info header. The header is included in PLAY_NOTIFY requests with the Notify-Reason of end-of-stream to provide RTP information about the end of the stream.

The header can carry the following parameters:

url: Indicates the stream URI which for which the following RTP parameters correspond, this URI MUST be the same used in the SETUP request for this media stream. Any relative URI MUST use the Request-URI as base URI. This parameter MUST be present.

ssrc: The Synchronization source (SSRC) that the RTP timestamp and sequence number provide applies to. This parameter MUST be present.

seq: Indicates the sequence number of the first packet of the stream that is direct result of the request. This allows clients to gracefully deal with packets when seeking. The client uses this value to differentiate packets that originated before the seek from packets that originated after the seek. Note that a client may not receive the packet with the expressed sequence number, and instead packets with a higher sequence number, due to packet loss or reordering. This parameter is RECOMMENDED to be present.

rtptime: MUST indicate the RTP timestamp value corresponding to the start time value in the Range response header, or if not explicitly given the implied start point. The client uses this value to calculate the mapping of RTP time to NPT or other media timescale. This parameter SHOULD be present to ensure inter-media synchronization is achieved. There exist no requirement that any received RTP packet will have the same RTP timestamp value as the one in the parameter used to establish synchronization.

A mapping from RTP timestamps to NTP timestamps (wallclock) is available via RTCP. However, this information is not sufficient to generate a mapping from RTP timestamps to media clock time (NPT, etc.). Furthermore, in order to ensure that this information is available at the necessary time (immediately at startup or after a seek), and that it is delivered reliably, this mapping is placed in the RTSP control channel.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1438
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

In order to compensate for drift for long, uninterrupted presentations, RTSP clients should additionally map NPT to NTP, using initial RTCP sender reports to do the mapping, and later reports to check drift against the mapping.

Example:

```
Range:npt=3.25-15
RTP-Info:url="rtsp://example.com/foo/audio" ssrc=0A13C760:seq=45102;
         rtptime=12345678,url="rtsp://example.com/foo/video"
         ssrc=9A9DE123:seq=30211;rtptime=29567112
```

Lets assume that Audio uses a 16kHz RTP timestamp clock and Video a 90kHz RTP timestamp clock. Then the media synchronization is depicted in the following way.

```
NPT    3.0---3.1---3.2-X-3.3---3.4---3.5---3.6
Audio               PA A
Video                  V    PV
```

X: NPT time value = 3.25, from Range header.
A: RTP timestamp value for Audio from RTP-Info header (12345678).
V: RTP timestamp value for Video from RTP-Info header (29567112).
PA: RTP audio packet carrying an RTP timestamp of 12344878. Which corresponds to NPT = (12344878 - A) / 16000 + 3.25 = 3.2
PV: RTP video packet carrying an RTP timestamp of 29573412. Which corresponds to NPT = (29573412 - V) / 90000 + 3.25 = 3.32

16.44. Scale

A scale value of 1 indicates normal play at the normal forward viewing rate. If not 1, the value corresponds to the rate with respect to normal viewing rate. For example, a ratio of 2 indicates twice the normal viewing rate ("fast forward") and a ratio of 0.5 indicates half the normal viewing rate. In other words, a ratio of 2 has content time increase at twice the playback time. For every second of elapsed (wallclock) time, 2 seconds of content time will be delivered. A negative value indicates reverse direction. For certain media transports this may require certain considerations to work consistent, see Appendix C.1 for description on how RTP handles this.

The transmitted data rate SHOULD NOT be changed by selection of a different scale value. The resulting bit-rate should be reasonably close to the nominal bit-rate of the content for Scale = 1. The server has to actively manipulate the data when needed to meet the bitrate constraints. Implementation of scale changes depends on the server and media type. For video, a server may, for example, deliver only key frames or selected frames. For audio, it may time-scale the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1439
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

audio while preserving pitch or, less desirably, deliver fragments of audio, or completely mute the audio.

The server and content may restrict the range of scale values that it supports. The supported values are indicated by the Media-Properties header (Section 16.28). The client SHOULD only indicate values indicated to be supported. However, as the values may change as the content progresses a requested value may no longer be valid when the request arrives. Thus, a non-supported value in a request does not generate an error, only forces the server to choose the closest value. The response MUST always contain the actual scale value chosen by the server.

If the server does not implement the possibility to scale, it will not return a Scale header. A server supporting Scale operations for PLAY MUST indicate this with the use of the "play.scale" feature-tag.

When indicating a negative scale for a reverse playback, the Range header MUST indicate a decreasing range as described in Section 16.38.

Example of playing in reverse at 3.5 times normal rate:

```
Scale: -3.5
Range: npt=15-10
```

16.45. Seek-Style

When a client sends a PLAY request with a Range header to perform a random access to the media, the client does not know if the server will pick the first media samples or the first random access point prior to the request range. Depending on use case, the client may have a strong preference. To express this preference and provide the client with information on how the server actually acted on that preference the Seek-Style header is defined.

Seek-Style is a general header that MAY be included in any PLAY request to indicate the client's preference for any media stream that has random access properties. The server MUST always include the header in any PLAY response for media with random access properties to indicate what policy was applied. A server that receives a unknown Seek-Style policy MUST ignore it and select the server default policy. A client receiving an unknown policy MUST ignore it and use the Range header and any media synchronization information as basis to determine what the server did.

This specification defines the following seek policies that may be requested (see also Section Section 4.9.1):

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1440
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RAP: Random Access Point (RAP) is the behavior of requesting the server to locate the closest previous random access point that exist in the media aggregate and deliver from that. By requesting a RAP, media quality will be the best possible as all media will be delivered from a point where full media state can be established in the media decoder.

CoRAP: Conditional Random Access Point (CoRAP) is a variant of the above RAP behavior. This policy is primarily intended for cases where there are larger distance between the random access points in the media. CoRAP is conditioned on that there is a Random Access Point closer to the requested start point than to the current pause point. This policy assumes that the media state existing prior to the pause is usable if delivery is continued. If the client or server knows that this is not the fact the RAP policy should be used. In other words: in most cases when the client requests a start point prior to the current pause point, a valid decoding dependency chain from the media delivered prior to the pause and to the requested media unit will not exist. If the server searched to a random access point the server MUST return the CoRAP policy in the Seek-Style header and adjust the Range header to reflect the position of the picked RAP. In case the random access point is further away and the server selects to continue from the current pause point it MUST include the "Next" policy in the Seek-Style header and adjust the Range header start point to the current pause point.

First-Prior: The first-prior policy will start delivery with the media unit that has a playout time first prior to the requested time. For discrete media that would only include media units that would still be rendered at the request time. For continuous media that is media that will be render during the requested start time of the range.

Next: The next media units after the provided start time of the range. For continuous framed media that would mean the first next frame after the provided time. For discrete media the first unit that is to be rendered after the provided time. The main usage is for this case is when the client knows it has all media up to a certain point and would like to continue delivery so that a complete non-interrupted media playback can be achieved. Example of such scenarios include switching from a broadcast/multicast delivery to a unicast based delivery. This policy MUST only be used on the client's explicit request.

Please note that these expressed preferences exist for optimizing the startup time or the media quality. The "Next" policy breaks the normal definition of the Range header to enable a client to request

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1441
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

media with minimal overlap, although some may still occur for aggregated sessions. RAP and First-Prior both fulfill the requirement of providing media from the requested range and forward. However, unless RAP is used, the media quality for many media codecs using predictive methods can be severely degraded unless additional data is available as, for example, already buffered, or through other side channels.

16.46. Server

The Server response-header field contains information about the software used by the origin server to handle the request. The field can contain multiple product tokens and comments identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application.

Example:
Server: PhonyServer/1.0

If the response is being forwarded through a proxy, the proxy application MUST NOT modify the Server response-header. Instead, it SHOULD include a Via field (Section 16.56).

16.47. Session

The Session request-header and response-header field identifies an RTSP session. An RTSP session is created by the server as a result of a successful SETUP request and in the response the session identifier is given to the client. The RTSP session exist until destroyed by a TEARDOWN, REDIRECT or timed out by the server.

The session identifier is chosen by the server (see Section 4.3) and MUST be returned in the SETUP response. Once a client receives a session identifier, it MUST be included in any request related to that session. This means that the Session header MUST be included in a request using the following methods: PLAY, PAUSE, and TEARDOWN, and MAY be included in SETUP, OPTIONS, SET_PARAMETER, GET_PARAMETER, and REDIRECT, and MUST NOT be included in DESCRIBE. In an RTSP response the session header MUST be included in methods, SETUP, PLAY, and PAUSE, and MAY be included in methods, TEARDOWN, and REDIRECT, and if included in the request of the following methods it MUST also be included in the response, OPTIONS, GET_PARAMETER, and SET_PARAMETER, and MUST NOT be included in DESCRIBE.

Note that a session identifier identifies an RTSP session across transport sessions or connections. RTSP requests for a given session can use different URIs (Presentation and media URIs). Note, that

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1442
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

there are restrictions depending on the session which URIs that are acceptable for a given method. However, multiple "user" sessions for the same URI from the same client will require use of different session identifiers.

The session identifier is needed to distinguish several delivery requests for the same URI coming from the same client.

The response 454 (Session Not Found) MUST be returned if the session identifier is invalid.

The header MAY include the session timeout period. If not explicitly provided this value is set to 60 seconds. As this affects how often session keep-alives are needed values smaller than 30 seconds are not recommended. However, larger than default values can be useful in applications of RTSP that have inactive but established sessions for longer time periods.

60 seconds was chosen as session timeout value due to: Resulting in not to frequent keep-alive messages and having low sensitivity to variations in request response timing. If one reduces the timeout value to below 30 seconds the corresponding request response timeout becomes a significant part of the session timeout. 60 seconds also allows for reasonably rapid recovery of committed server resources in case of client failure.

16.48. Speed

The Speed request-header field requests the server to deliver specific amounts of nominal media time per unit of delivery time, contingent on the server's ability and desire to serve the media stream at the given speed. The client requests the delivery speed to be within a given range with an lower and upper bound. The server SHALL deliver at the highest possible speed within the range, but not faster than the upper-bound, for which the underlying network path can support the resulting transport data rates. As long as any speed value within the given range can be provided the server SHALL NOT modify the media quality. Only if the server is unable to delivery media at the speed value provided by the lower bound shall it reduce the media quality.

Implementation of the Speed functionality by the server is OPTIONAL. The server can indicate its support through a feature-tag, play.speed. The lack of a Speed header in the response is an indication of lack of support of this functionality.

The speed parameter values are expressed as a positive decimal value, e.g., a value of 2.0 indicates that data is to be delivered twice as

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1443
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

fast as normal. A speed value of zero is invalid. The range is specified in the form "lower bound - upper bound". The lower bound value may be smaller or equal to the upper bound. All speeds may not be possible to support. Therefore the server MAY modify the requested values to the closest supported. The actual supported speed MUST be included in the response. Note, however, that the use cases may vary and that Speed value ranges such as 0.7 - 0.8, 0.3-2.0, 1.0-2.5, 2.5-2.5 all have their usage.

Example:

```
Speed: 1.0-2.5
```

Use of this header changes the bandwidth used for data delivery. It is meant for use in specific circumstances where delivery of the presentation at a higher or lower rate is desired. The main use cases are buffer operations or local scale operations. Implementors should keep in mind that bandwidth for the session may be negotiated beforehand (by means other than RTSP), and therefore re-negotiation may be necessary. To perform Speed operations the server needs to ensure that the network path can support the resulting bit-rate. Thus the media transport needs to support feedback so that the server can react and adapt to the available bitrate.

16.49. Supported

The Supported header enumerates all the extensions supported by the client or server using feature tags. The header carries the extensions supported by the message sending client or server. The Supported header MAY be included in any request. When present in a request, the receiver MUST respond with its corresponding Supported header. Note that the supported headers is also included in 4xx and 5xx responses.

The Supported header contains a list of feature-tags, described in Section 4.7, that are understood by the client or server.

Example:

```
C->S:  OPTIONS rtsp://example.com/ RTSP/2.0
       Supported: foo, bar, blech
       User-Agent: PhonyClient/1.2

S->C:  RTSP/2.0 200 OK
       Supported: bar, blech, baz
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1444
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.50. Terminate-Reason

The Terminate-Reason request header allows the server when sending a REDIRECT or TEARDOWN request to provide a reason for the session termination and any additional information. This specification identifies three reasons for Redirections and may be extended in the future:

Server-Admin: The server needs to be shutdown for some administrative reason.

Session-Timeout: A client's session is kept alive for extended periods of time and the server has determined that it needs to reclaim the resources associated with this session.

Internal-Error An internal error that is impossible to recover from has occurred forcing the server to terminate the session.

The Server may provide additional parameters containing information around the redirect. This specification defines the following ones.

time: Provides a wallclock time when the server will stop provide any service.

user-msg: An UTF-8 text string with a message from the server to the user. This message SHOULD be displayed to the user.

16.51. Timestamp

The Timestamp general-header describes when the agent sent the request. The value of the timestamp is of significance only to the agent and may use any timescale. The responding agent MUST echo the exact same value and MAY, if it has accurate information about this, add a floating point number indicating the number of seconds that has elapsed since it has received the request. The timestamp can be used by the agent to compute the round-trip time to the responding agent so that it can adjust the timeout value for retransmissions when running over a unreliable protocol. It also resolves retransmission ambiguities for unreliable transport of RTSP.

Note that the present specification provides only for reliable transport of RTSP messages. The Timestamp general-header is specified in case the protocol is extended in the future to use unreliable transport.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1445
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.52. Transport

The Transport request and response header indicates which transport protocol is to be used and configures its parameters such as destination address, compression, multicast time-to-live and destination port for a single stream. It sets those values not already determined by a presentation description.

A Transport request header MAY contain a list of transport options acceptable to the client, in the form of multiple transport specification entries. Transport specifications are comma separated, listed in decreasing order of preference. Parameters may be added to each transport specification, separated by a semicolon. The server MUST return a Transport response-header in the response to indicate the values actually chosen if any. If the transport specification is not supported, no transport header is returned and the request MUST be responded using the status code 461 (Unsupported Transport) (Section 15.4.26). In case more than one transport specification was present in the request, the server MUST return the single (transport-spec) which was actually chosen, if any. The number of transport-spec entries is expected to be limited as the client will get guidance on what configurations that are possible from the presentation description.

The Transport header MAY also be used in subsequent SETUP requests to change transport parameters. A server MAY refuse to change parameters of an existing stream.

A transport specification may only contain one of any given parameter within it. Parameters MAY be given in any order. Additionally, it may only contain either of the unicast or the multicast transport type parameter. All parameters need to be understood in a transport specification, if not, the transport specification MUST be ignored. RTSP proxies of any type that uses or modifies the transport specification, e.g. access proxy or security proxy, MUST remove specifications with unknown parameters before forwarding the RTSP message. If that result in no remaining transport specification the proxy shall send a 461 (Unsupported Transport) (Section 15.4.26) response without any Transport header.

> The Transport header is restricted to describing a single media stream. (RTSP can also control multiple streams as a single entity.) Making it part of RTSP rather than relying on a multitude of session description formats greatly simplifies designs of firewalls.

The general syntax for the transport specifier is a list of slash separated tokens:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1446
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Value1/Value2/Value3...
Which for RTP transports take the form:
RTP/profile/lower-transport.

The default value for the "lower-transport" parameters is specific to the profile. For RTP/AVP, the default is UDP.

There are two different methods for how to specify where the media should be delivered for unicast transport:

dest_addr: The presence of this parameter and its values indicates the destination address or addresses (host address and port pairs for IP flows) necessary for the media transport.

No dest_addr: The lack of the dest_addr parameter indicates that the server MUST send media to same address for which the RTSP messages originates.

The choice of method for indicating where the media is to be delivered depends on the use case. In some case the only allowed method will be to use no explicit address indication and have the server deliver media to the source of the RTSP messages.

For Multicast there is several methods for specifying addresses but they are different in how they work compared with unicast:

dest_addr with client picked address: The address and relevant parameters like TTL (scope) for the actual multicast group to deliver the media to. There are security implications (Section 21) with this method that needs to be addressed if using this method because a RTSP server can be used as a DoS attacker on a existing multicast group.

dest_addr using Session Description Information: The information included in the transport header can all be coming from the session description, e.g. the SDP c= and m= line. This mitigates some of the security issues of the previous methods as it is the session provider that picks the multicast group and scope. The client MUST include the information if it is available in the session description.

No dest_addr: The behavior when no explicit multicast group is present in a request is not defined.

An RTSP proxy will need to take care. If the media is not desired to be routed through the proxy, the proxy will need to introduce the destination indication.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1447
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Below are the configuration parameters associated with transport:

General parameters:

unicast / multicast: This parameter is a mutually exclusive indication of whether unicast or multicast delivery will be attempted. One of the two values MUST be specified. Clients that are capable of handling both unicast and multicast transmission needs to indicate such capability by including two full transport-specs with separate parameters for each.

layers: The number of multicast layers to be used for this media stream. The layers are sent to consecutive addresses starting at the dest_addr address. If the parameter is not included, it defaults to a single layer.

dest_addr: A general destination address parameter that can contain one or more address specifications. Each combination of protocol/profile/lower transport needs to have the format and interpretation of its address specification defined. For RTP/AVP/UDP and RTP/AVP/TCP, the address specification is a tuple containing a host address and port. Note, only a single destination parameter per transport spec is intended. The usage of multiple destination to distribute a single media to multiple entities is unspecified.

The client originating the RTSP request MAY specify the destination address of the stream recipient with the host address part of the tuple. When the destination address is specified, the recipient may be a different party than the originator of the request. To avoid becoming the unwitting perpetrator of a remote-controlled denial-of-service attack, a server MUST perform security checks (see Section 21.1) and SHOULD log such attempts before allowing the client to direct a media stream to a recipient address not chosen by the server. Implementations cannot rely on TCP as reliable means of client identification. If the server does not allow the host address part of the tuple to be set, it MUST return 463 (Destination Prohibited).

The host address part of the tuple MAY be empty, for example ":58044", in cases when only destination port is desired to be specified. Responses to requests including the Transport header with a dest_addr parameter SHOULD include the full destination address that is actually used by the server. The server MUST NOT remove address information present already in the request when responding unless the protocol requires it.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1448
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

src_addr: A general source address parameter that can contain one or more address specifications. Each combination of protocol/ profile/lower transport needs to have the format and interpretation of its address specification defined. For RTP/ AVP/UDP and RTP/AVP/TCP, the address specification is a tuple containing a host address and port.

This parameter MUST be specified by the server if it transmits media packets from another address than the one RTSP messages are sent to. This will allow the client to verify source address and give it a destination address for its RTCP feedback packets if RTP is used. The address or addresses indicated in the src_addr parameter SHOULD be used both for sending and receiving of the media streams data packets. The main reasons are threefold: First, indicating the port and source address(s) lets the receiver know where from the packets is expected to originate. Secondly, traversal of NATs are greatly simplified when traffic is flowing symmetrically over a NAT binding. Thirdly, certain NAT traversal mechanisms, needs to know to which address and port to send so called "binding packets" from the receiver to the sender, thus creating a address binding in the NAT that the sender to receiver packet flow can use.

This information may also be available through SDP. However, since this is more a feature of transport than media initialization, the authoritative source for this information should be in the SETUP response.

mode: The mode parameter indicates the methods to be supported for this session. Currently defined valid values are "PLAY". If not provided, the default is "PLAY". The "RECORD" value was defined in RFC 2326 and is in this specification unspecified but reserved. RECORD and other values may be specified in the future.

interleaved: The interleaved parameter implies mixing the media stream with the control stream in whatever protocol is being used by the control stream, using the mechanism defined in Section 14. The argument provides the channel number to be used in the $ statement and MUST be present. This parameter MAY be specified as a interval, e.g., interleaved=4-5 in cases where the transport choice for the media stream requires it, e.g. for RTP with RTCP. The channel number given in the request are only a guidance from the client to the server on what channel number(s) to use. The server MAY set any valid channel number in the response. The declared channel(s) are bi-directional, so both end-parties MAY send data on the given

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1449
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

channel. One example of such usage is the second channel used for RTCP, where both server and client sends RTCP packets on the same channel.

This allows RTP/RTCP to be handled similarly to the way that it is done with UDP, i.e., one channel for RTP and the other for RTCP.

MIKEY: This parameter is used in conjunction with transport specifications that can utilize MIKEY for security context establishment. So far only the SRTP based RTP profiles SAVP and SAVPF can utilize MIKEY and this is defined in Appendix C.1.4.1. This parameter can be included both in request and response messages. The binary MIKEY message SHALL be BASE64 [RFC4648] encoded before being included in the value part of the parameter.

Multicast-specific:

ttl: multicast time-to-live for IPv4. When included in requests the value indicate the TTL value that the client request the server to use. In a response, the value actually being used by the server is returned. A server will need to consider what values that are reasonable and also the authority of the user to set this value. Corresponding functions are not needed for IPv6 as the scoping is part of the address.

RTP-specific:

These parameters are MAY only be used if the media transport protocol is RTP.

ssrc: The ssrc parameter, if included in a SETUP response, indicates the RTP SSRC [RFC3550] value(s) that will be used by the media server for RTP packets within the stream. It is expressed as an eight digit hexadecimal value.

The ssrc parameter MUST NOT be specified in requests. The functionality of specifying the ssrc parameter in a SETUP request is deprecated as it is incompatible with the specification of RTP in RFC 3550[RFC3550]. If the parameter is included in the Transport header of a SETUP request, the server MAY ignore it, and choose appropriate SSRCs for the stream. The server MAY set the ssrc parameter in the Transport header of the response.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1450
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RTCP-mux: Use to negotiate the usage of RTP and RTCP multiplexing [RFC5761] on a single underlying transport stream / flow. The presence of this parameter in a SETUP request indicates the clients support and requires the server to use RTP and RTCP multiplexing. The client SHALL only include one transport stream in the Transport header specification. To provide the server with a choice between using RTP/RTCP multiplexing or not, two different transport header specifications must be included.

The parameters setup and connection defined below MAY only be used if the media transport protocol of the lower-level transport is connection-oriented (such as TCP). However, these parameters MUST NOT be used when interleaving data over the RTSP control connection.

setup: Clients use the setup parameter on the Transport line in a SETUP request, to indicate the roles it wishes to play in a TCP connection. This parameter is adapted from [RFC4145]. We discuss the use of this parameter in RTP/AVP/TCP non-interleaved transport in Appendix C.2.2; the discussion below is limited to syntactic issues. Clients may specify the following values for the setup parameter: ["active":] The client will initiate an outgoing connection. ["passive":] The client will accept an incoming connection. ["actpass":] The client is willing to accept an incoming connection or to initiate an outgoing connection.

If a client does not specify a setup value, the "active" value is assumed.

In response to a client SETUP request where the setup parameter is set to "active", a server's 2xx reply MUST assign the setup parameter to "passive" on the Transport header line.

In response to a client SETUP request where the setup parameter is set to "passive", a server's 2xx reply MUST assign the setup parameter to "active" on the Transport header line.

In response to a client SETUP request where the setup parameter is set to "actpass", a server's 2xx reply MUST assign the setup parameter to "active" or "passive" on the Transport header line.

Note that the "holdconn" value for setup is not defined for RTSP use, and MUST NOT appear on a Transport line.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1451
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

connection: Clients use the setup parameter on the Transport line in a SETUP request, to indicate the SETUP request prefers the reuse of an existing connection between client and server (in which case the client sets the "connection" parameter to "existing"), or that the client requires the creation of a new connection between client and server (in which cast the client sets the "connection" parameter to "new"). Typically, clients use the "new" value for the first SETUP request for a URL, and "existing" for subsequent SETUP requests for a URL.

If a client SETUP request assigns the "new" value to "connection", the server response MUST also assign the "new" value to "connection" on the Transport line.

If a client SETUP request assigns the "existing" value to "connection", the server response MUST assign a value of "existing" or "new" to "connection" on the Transport line, at its discretion.

The default value of "connection" is "existing", for all SETUP requests (initial and subsequent).

The combination of transport protocol, profile and lower transport needs to be defined. A number of combinations are defined in the Appendix C.

Below is a usage example, showing a client advertising the capability to handle multicast or unicast, preferring multicast. Since this is a unicast-only stream, the server responds with the proper transport parameters for unicast.

```
C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/2.0
      CSeq: 302
      Transport: RTP/AVP;multicast;mode="PLAY",
          RTP/AVP;unicast;dest_addr="192.0.2.5:3456"/
          "192.0.2.5:3457";mode="PLAY"
      Accept-Ranges: NPT, SMPTE, UTC
      User-Agent: PhonyClient/1.2

S->C: RTSP/2.0 200 OK
      CSeq: 302
      Date: Thu, 23 Jan 1997 15:35:06 GMT
      Session: 47112344
      Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:3456"/
         "192.0.2.5:3457";src_addr="192.0.2.224:6256"/
         "192.0.2.224:6257";mode="PLAY"
      Accept-Ranges: NPT
      Media-Properties: Random-Access=0.6, Dynamic,
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1452
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Time-Limited=20081128T165900

16.53. Unsupported

The Unsupported response-header lists the features not supported by the responding RTSP agent. In the case where the feature was specified via the Proxy-Require field (Section 16.35), if there is a proxy on the path between the client and the server, the proxy MUST send a response message with a status code of 551 (Option Not Supported). The request MUST NOT be forwarded.

See Section 16.41 for a usage example.

16.54. User-Agent

The User-Agent general-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests. The field can contain multiple product tokens and comments identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.

Example:
User-Agent: PhonyClient/1.2

16.55. Vary

The Vary field value indicates the set of request-header fields that fully determines, while the response is fresh, whether a cache is permitted to use the response to reply to a subsequent request without revalidation. For uncacheable or stale responses, the Vary field value advises the user agent about the criteria that were used to select the representation. A Vary field value of "*" implies that a cache cannot determine from the request headers of a subsequent request whether this response is the appropriate representation.

An RTSP server SHOULD include a Vary header field with any cacheable response that is subject to server-driven negotiation. Doing so allows a cache to properly interpret future requests on that resource and informs the user agent about the presence of negotiation on that resource. A server MAY include a Vary header field with a non-cacheable response that is subject to server-driven negotiation, since this might provide the user agent with useful information about the dimensions over which the response varies at the time of the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1453
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

response.

A Vary field value consisting of a list of field-names signals that the representation selected for the response is based on a selection algorithm which considers ONLY the listed request-header field values in selecting the most appropriate representation. A cache MAY assume that the same selection will be made for future requests with the same values for the listed field names, for the duration of time for which the response is fresh.

The field-names given are not limited to the set of standard request-header fields defined by this specification. Field names are case-insensitive.

A Vary field value of "*" signals that unspecified parameters not limited to the request-headers (e.g., the network address of the client), play a role in the selection of the response representation. The "*" value MUST NOT be generated by a proxy server; it may only be generated by an origin server.

16.56. Via

The Via general-header field MUST be used by proxies to indicate the intermediate protocols and recipients between the user agent and the server on requests, and between the origin server and the client on responses. The field is intended to be used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of all senders along the request/response chain.

Multiple Via field values represents each proxy that has forwarded the message. Each recipient MUST append its information such that the end result is ordered according to the sequence of forwarding applications.

Proxies (e.g., Access Proxy or Translator Proxy) SHOULD NOT, by default, forward the names and ports of hosts within the private/protected region. This information SHOULD only be propagated if explicitly enabled. If not enabled, the via-received of any host behind the firewall/NAT SHOULD be replaced by an appropriate pseudonym for that host.

For organizations that have strong privacy requirements for hiding internal structures, a proxy MAY combine an ordered subsequence of Via header field entries with identical sent-protocol values into a single such entry. Applications MUST NOT combine entries which have different received-protocol values.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1454
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

16.57. WWW-Authenticate

The WWW-Authenticate response-header field MUST be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI.

The HTTP access authentication process is described in [RFC2617]. User agents are advised to take special care in parsing the WWW-Authenticate field value as it might contain more than one challenge, or if more than one WWW-Authenticate header field is provided, the contents of a challenge itself can contain a comma-separated list of authentication parameters.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1455
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

17. Proxies

RTSP Proxies are RTSP agents that sit in between a client and a server. A proxy can take on both the role as a client and as server depending on what it tries to accomplish. Proxies are also introduced for several different reasons and the below are often combined.

Caching Proxy: This type of proxy is used to reduce the workload on servers and connections. By caching the description and media streams, i.e., the presentation, the proxy can serve a client with content, but without requesting it from the server once it has been cached and has not become stale. See the caching Section 18. This type of proxy is also expected to understand RTSP end-point functionality, i.e., functionality identified in the Require header in addition to what Proxy-Require demands.

Translator Proxy: This type of proxy is used to ensure that an RTSP client get access to servers and content on an external network or using content encodings not supported by the client. The proxy performs the necessary translation of addresses, protocols or encodings. This type of proxy is expected to also understand RTSP end-point functionality, i.e. functionality identified in the Require header in addition to what Proxy-Require demands.

Access Proxy: This type of proxy is used to ensure that a RTSP client get access to servers on an external network. Thus this proxy is placed on the border between two domains, e.g. a private address space and the public Internet. The proxy performs the necessary translation, usually addresses. This type of proxies are required to redirect the media to themselves or a controlled gateway that perform the translation before the media can reach the client.

Security Proxy: This type of proxy is used to help facilitate security functions around RTSP. For example when having a firewalled network, the security proxy request that the necessary pinholes in the firewall is opened when a client in the protected network want to access media streams on the external side. This proxy can also limit the clients access to certain type of content. This proxy can perform its function without redirecting the media between the server and client. However, in deployments with private address spaces this proxy is likely to be combined with the access proxy. Anyway, the functionality of this proxy is usually closely tied into understand all aspects of the media transport.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1456
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Auditing Proxy: RTSP proxies can also provide network owners with a logging and audit point for RTSP sessions, e.g. for corporations that tracks their employees usage of the network. This type of proxy can perform its function without inserting itself or any other node in the media transport. This proxy type can also accept unknown methods as it doesn't interfere with the clients' requests.

All type of proxies can be used also when using secured communication with TLS as RTSP 2.0 allows the client to approve certificate chains used for connection establishment from a proxy, see Section 19.3.2. However, that trust model may not be suitable for all type of deployment. In those cases, the secured sessions do by-pass of the proxies.

Access proxies SHOULD NOT be used in equipment like NATs and firewalls that aren't expected to be regularly maintained, like home or small office equipment. In these cases it is better to use the NAT traversal procedures defined for RTSP 2.0 [I-D.ietf-mmusic-rtsp-nat]. The reason for these recommendations is that any extensions of RTSP resulting in new media transport protocols or profiles, new parameters etc may fail in a proxy that isn't maintained. This would impede RTSP's future development and usage.

17.1. Proxies and Protocol Extensions

The existence of proxies must always be considered when developing new RTSP extensions. Most types of proxies will need to implement any new method to operate correctly in the presence of that extension. New headers can be introduced and will not be blocked by older proxies. However, it is important to consider if this header and its function is required to be understood by the proxy or can be forwarded. If the header needs to be understood a feature-tag representing the functionality needs to be included in the Proxy-Require header. Below are guidelines for analysis if the header needs to be understood. The transport header and its parameters also shows that headers that are extensible and require correct interpretation in the proxy also require handling rules.

Whether a proxy needs to understand a header is not easy to determine, as they serve a broad variety of functions. When evaluating if a header needs to be understood, one can divide the functionality into three main categories:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1457
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Media modifying: The caching and translator proxies are modifying the actual media and therefore needs to understand also request directed to the server that affects how the media is rendered. Thus, this type of proxies needs to also understand the server side functionality.

Transport modifying: The access and the security proxy both need to understand how the transport is performed, either for opening pinholes or to translate the outer headers, e.g. IP and UDP.

Non-modifying: The audit proxy is special in that it do not modify the messages in other ways than to insert the Via header. That makes it possible for this type to forward RTSP message that contains different type of unknown methods, headers or header parameters.

Based on the above classification, one should evaluate if the new functionality requires the Transport modifying type of proxies to understand it or not.

17.2. Multiplexing and Demultiplexing of Messages

RTSP proxies may have to multiplex multiple RTSP sessions from their clients towards RTSP servers. This requires that RTSP requests from multiple clients are multiplexed onto a common connection for requests outgoing to a RTSP server and on the way back the responses are demultiplexed from the server to per client responses. On the protocol level this requires that request and response messages are handled in both ways, requiring that there is a mechanism to correlated what request/response pair exchanged between proxy and server is mapped to what client (or client request).

This multiplexing of requests and demultiplexing of responses is done by using the CSeq header field (see Section 16.19). The proxy has to rewrite the CSeq in requests to the server and responses from the server and remember what CSeq is mapped to what client.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1458
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

18. Caching

In HTTP, response-request pairs are cached. RTSP differs significantly in that respect. Responses are not cacheable, with the exception of the presentation description returned by DESCRIBE. (Since the responses for anything but DESCRIBE and GET_PARAMETER do not return any data, caching is not really an issue for these requests.) However, it is desirable for the continuous media data, typically delivered out-of-band with respect to RTSP, to be cached, as well as the session description.

On receiving a SETUP or PLAY request, a proxy ascertains whether it has an up-to-date copy of the continuous media content and its description. It can determine whether the copy is up-to-date by issuing a SETUP or DESCRIBE request, respectively, and comparing the Last-Modified header with that of the cached copy. If the copy is not up-to-date, it modifies the SETUP transport parameters as appropriate and forwards the request to the origin server. Subsequent control commands such as PLAY or PAUSE then pass the proxy unmodified. The proxy delivers the continuous media data to the client, while possibly making a local copy for later reuse. The exact allowed behavior of the cache is given by the cache-response directives described in Section 16.10. A cache MUST answer any DESCRIBE requests if it is currently serving the stream to the requester, as it is possible that low-level details of the stream description may have changed on the origin-server.

Note that an RTSP cache, is of the "cut-through" variety. Rather than retrieving the whole resource from the origin server, the cache simply copies the streaming data as it passes by on its way to the client. Thus, it does not introduce additional latency.

To the client, an RTSP proxy cache appears like a regular media server, to the media origin server like a client. Just as an HTTP cache has to store the content type, content language, and so on for the objects it caches, a media cache has to store the presentation description. Typically, a cache eliminates all transport-references (that is, e.g. multicast information) from the presentation description, since these are independent of the data delivery from the cache to the client. Information on the encodings remains the same. If the cache is able to translate the cached media data, it would create a new presentation description with all the encoding possibilities it can offer.

18.1. Validation Model

When a cache has a stale entry that it would like to use as a response to a client's request, it first has to check with the origin

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1459
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

server (or possibly an intermediate cache with a fresh response) to see if its cached entry is still usable. We call this "validating" the cache entry. Since we do not want to have to pay the overhead of retransmitting the full response if the cached entry is good, and we do not want to pay the overhead of an extra round trip if the cached entry is invalid, the RTSP protocol supports the use of conditional methods.

The key protocol features for supporting conditional methods are those concerned with "cache validators." When an origin server generates a full response, it attaches some sort of validator to it, which is kept with the cache entry. When a client (user agent or proxy cache) makes a conditional request for a resource for which it has a cache entry, it includes the associated validator in the request.

The server then checks that validator against the current validator for the requested resource, and, if they match (see Section 18.1.3), it responds with a special status code (usually, 304 (Not Modified)) and no message body. Otherwise, it returns a full response (including message body). Thus, we avoid transmitting the full response if the validator matches, and we avoid an extra round trip if it does not match.

In RTSP, a conditional request looks exactly the same as a normal request for the same resource, except that it carries a special header (which includes the validator) that implicitly turns the method (usually DESCRIBE or SETUP) into a conditional.

The protocol includes both positive and negative senses of cache-validating conditions. That is, it is possible to request either that a method be performed if and only if a validator matches or if and only if no validators match.

> Note: a response that lacks a validator may still be cached, and served from cache until it expires, unless this is explicitly prohibited by a cache-control directive (see Section 16.10). However, a cache cannot do a conditional retrieval if it does not have a validator for the resource, which means it will not be refreshable after it expires.

Media streams that are being adapted based on the transport capacity between the server and the cache makes caching more difficult. A server needs to consider how it views caching of media streams that it adapts and potentially instruct any caches to not cache such streams.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1460
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

18.1.1. Last-Modified Dates

The Last-Modified header (Section 16.26) value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the content has not been modified since the Last-Modified value.

18.1.2. Message Body Tag Cache Validators

The MTag response-header field value, an message body tag, provides for an "opaque" cache validator. This might allow more reliable validation in situations where it is inconvenient to store modification dates, where the one-second resolution of RTSP-date values is not sufficient, or where the origin server wishes to avoid certain paradoxes that might arise from the use of modification dates.

Message body tags are described in Section 5.3

18.1.3. Weak and Strong Validators

Since both origin servers and caches will compare two validators to decide if they represent the same or different entities, one normally would expect that if the message body (i.e., the presentation description) or any associated message body headers changes in any way, then the associated validator would change as well. If this is true, then we call this validator a "strong validator." We call message body (i.e., the presentation description) or any associated message body headers an entity for a better understanding.

However, there might be cases when a server prefers to change the validator only on semantically significant changes, and not when insignificant aspects of the entity change. A validator that does not always change when the resource changes is a "weak validator."

Message body tags are normally "strong validators," but the protocol provides a mechanism to tag an message body tag as "weak." One can think of a strong validator as one that changes whenever the bits of an entity changes, while a weak value changes whenever the meaning of an entity changes. Alternatively, one can think of a strong validator as part of an identifier for a specific entity, while a weak validator is part of an identifier for a set of semantically equivalent entities.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1461
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Note: One example of a strong validator is an integer that is incremented in stable storage every time an entity is changed.

An entity's modification time, if represented with one-second resolution, could be a weak validator, since it is possible that the resource might be modified twice during a single second.

Support for weak validators is optional. However, weak validators allow for more efficient caching of equivalent objects.

A "use" of a validator is either when a client generates a request and includes the validator in a validating header field, or when a server compares two validators.

Strong validators are usable in any context. Weak validators are only usable in contexts that do not depend on exact equality of an entity. For example, either kind is usable for a conditional DESCRIBE of a full entity. However, only a strong validator is usable for a sub-range retrieval, since otherwise the client might end up with an internally inconsistent entity.

Clients MAY issue DESCRIBE requests with either weak validators or strong validators. Clients MUST NOT use weak validators in other forms of request.

The only function that the RTSP protocol defines on validators is comparison. There are two validator comparison functions, depending on whether the comparison context allows the use of weak validators or not:

- The strong comparison function: in order to be considered equal, both validators MUST be identical in every way, and both MUST NOT be weak.

- The weak comparison function: in order to be considered equal, both validators MUST be identical in every way, but either or both of them MAY be tagged as "weak" without affecting the result.

An message body tag is strong unless it is explicitly tagged as weak.

A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:

- The validator is being compared by an origin server to the actual current validator for the entity and,

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1462
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- That origin server reliably knows that the associated entity did not change twice during the second covered by the presented validator.

OR

- The validator is about to be used by a client in an If-Modified-Since, because the client has a cache entry for the associated entity, and
- That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- The presented Last-Modified time is at least 60 seconds before the Date value.

OR

- The validator is being compared by an intermediate cache to the validator stored in its cache entry for the entity, and
- That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- The presented Last-Modified time is at least 60 seconds before the Date value.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-Modified values are generated from different clocks, or at somewhat different times during the preparation of the response. An implementation MAY use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

If a client wishes to perform a sub-range retrieval on a value for which it has only a Last-Modified time and no opaque validator, it MAY do this only if the Last-Modified time is strong in the sense described here.

18.1.4. Rules for When to Use Message Body Tags and Last-Modified Dates

We adopt a set of rules and recommendations for origin servers, clients, and caches regarding when various validator types ought to be used, and for what purposes.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1463
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RTSP origin servers:

- SHOULD send an message body tag validator unless it is not feasible to generate one.

- MAY send a weak message body tag instead of a strong message body tag, if performance considerations support the use of weak message body tags, or if it is unfeasible to send a strong message body tag.

- SHOULD send a Last-Modified value if it is feasible to send one, unless the risk of a breakdown in semantic transparency that could result from using this date in an If-Modified-Since header would lead to serious problems.

In other words, the preferred behavior for an RTSP origin server is to send both a strong message body tag and a Last-Modified value.

In order to be legal, a strong message body tag MUST change whenever the associated entity value changes in any way. A weak message body tag SHOULD change whenever the associated entity changes in a semantically significant way.

> Note: in order to provide semantically transparent caching, an origin server must avoid reusing a specific strong message body tag value for two different entities, or reusing a specific weak message body tag value for two semantically different entities. Cache entries might persist for arbitrarily long periods, regardless of expiration times, so it might be inappropriate to expect that a cache will never again attempt to validate an entry using a validator that it obtained at some point in the past.

RTSP clients:

- If an message body tag has been provided by the origin server, MUST use that message body tag in any cache-conditional request (using If- Match or If-None-Match).

- If only a Last-Modified value has been provided by the origin server, SHOULD use that value in non-subrange cache-conditional requests (using If-Modified-Since).

- If both an message body tag and a Last-Modified value have been provided by the origin server, SHOULD use both validators in cache-conditional requests.

An RTSP origin server, upon receiving a conditional request that includes both a Last-Modified date (e.g., in an If-Modified-Since

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1464
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

header) and one or more message body tags (e.g., in an If-Match, If-None-Match, or If-Range header field) as cache validators, MUST NOT return a response status of 304 (Not Modified) unless doing so is consistent with all of the conditional header fields in the request.

> Note: The general principle behind these rules is that RTSP servers and clients should transmit as much non-redundant information as is available in their responses and requests. RTSP systems receiving this information will make the most conservative assumptions about the validators they receive.

18.1.5. Non-validating Conditionals

The principle behind message body tags is that only the service author knows the semantics of a resource well enough to select an appropriate cache validation mechanism, and the specification of any validator comparison function more complex than byte-equality would open up a can of worms. Thus, comparisons of any other headers are never used for purposes of validating a cache entry.

18.2. Invalidation After Updates or Deletions

The effect of certain methods performed on a resource at the origin server might cause one or more existing cache entries to become non-transparently invalid. That is, although they might continue to be "fresh," they do not accurately reflect what the origin server would return for a new request on that resource.

There is no way for the RTSP protocol to guarantee that all such cache entries are marked invalid. For example, the request that caused the change at the origin server might not have gone through the proxy where a cache entry is stored. However, several rules help reduce the likelihood of erroneous behavior.

In this section, the phrase "invalidate an entity" means that the cache will either remove all instances of that entity from its storage, or will mark these as "invalid" and in need of a mandatory revalidation before they can be returned in response to a subsequent request.

Some RTSP methods MUST cause a cache to invalidate an entity. This is either the entity referred to by the Request-URI, or by the Location or Content-Location headers (if present). These methods are:

- DESCRIBE

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1465
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- SETUP

In order to prevent denial of service attacks, an invalidation based on the URI in a Location or Content-Location header MUST only be performed if the host part is the same as in the Request-URI.

A cache that passes through requests for methods it does not understand SHOULD invalidate any entities referred to by the Request-URI.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1466
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

19. Security Framework

The RTSP security framework consists of two high level components: the pure authentication mechanisms based on HTTP authentication, and the message transport protection based on TLS, which is independent of RTSP. Because of the similarity in syntax and usage between RTSP servers and HTTP servers, the security for HTTP is re-used to a large extent.

19.1. RTSP and HTTP Authentication

RTSP and HTTP share common authentication schemes, and thus follow the same usage guidelines as specified in[RFC2617] and also in [H15]. Servers SHOULD implement both basic and digest [RFC2617] authentication. Client MUST implement both basic and digest authentication [RFC2617] so that Server who requires the client to authenticate can trust that the capability is present.

It should be stressed that using the HTTP authentication alone does not provide full control message security. Therefore, in environments requiring tighter security for the control messages, TLS SHOULD be used, see Section 19.2.

19.2. RTSP over TLS

RTSP MUST follow the same guidelines with regards to TLS [RFC5246] usage as specified for HTTP, see [RFC2818]. RTSP over TLS is separated from unsecured RTSP both on URI level and port level. Instead of using the "rtsp" scheme identifier in the URI, the "rtsps" scheme identifier MUST be used to signal RTSP over TLS. If no port is given in a URI with the "rtsps" scheme, port 322 MUST be used for TLS over TCP/IP.

When a client tries to setup an insecure channel to the server (using the "rtsp" URI), and the policy for the resource requires a secure channel, the server MUST redirect the client to the secure service by sending a 301 redirect response code together with the correct Location URI (using the "rtsps" scheme). A user or client MAY upgrade a non secured URI to a secured by changing the scheme from "rtsp" to "rtsps". A server implementing support for "rtsps" MUST allow this.

It should be noted that TLS allows for mutual authentication (when using both server and client certificates). Still, one of the more common ways TLS is used is to only provide server side authentication (often to avoid client certificates). TLS is then used in addition to HTTP authentication, providing transport security and server authentication, while HTTP Authentication is used to authenticate the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1467
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

client.

RTSP includes the possibility to keep a TCP session up between the client and server, throughout the RTSP session lifetime. It may be convenient to keep the TCP session, not only to save the extra setup time for TCP, but also the extra setup time for TLS (even if TLS uses the resume function, there will be almost two extra round trips). Still, when TLS is used, such behavior introduces extra active state in the server, not only for TCP and RTSP, but also for TLS. This may increase the vulnerability to DoS attacks.

In addition to these recommendations, Section 19.3 gives further recommendations of TLS usage with proxies.

19.3. Security and Proxies

The nature of a proxy is often to act as a "man-in-the-middle", while security is often about preventing the existence of a "man-in-the-middle". This section provides clients with the possibility to use proxies even when applying secure transports (TLS) between the RTSP agents. The TLS proxy mechanism allows for server and proxy identification using certificates. However, the client can not be identified based on certificates. The client needs to select between using the procedure specified below or using a TLS connection directly (by-passing any proxies) to the server. The choice may be dependent on policies.

There are basically two categories of proxies, the transparent proxies (of which the client is not aware) and the non-transparent proxies (of which the client is aware). An infrastructure based on proxies requires that the trust model is such that both client and servers can trust the proxies to handle the RTSP messages correctly. To be able to trust a proxy, the client and server also needs to be aware of the proxy. Hence, transparent proxies cannot generally be seen as trusted and will not work well with security (unless they work only at transport layer). In the rest of this section any reference to proxy will be to a non-transparent proxy, which inspects or manipulate the RTSP messages.

HTTP Authentication is built on the assumption of proxies and can provide user-proxy authentication and proxy-proxy/server authentication in addition to the client-server authentication.

When TLS is applied and a proxy is used, the client will connect to the proxy's address when connecting to any RTSP server. This implies that for TLS, the client will authenticate the proxy server and not the end server. Note that when the client checks the server certificate in TLS, it MUST check the proxy's identity (URI or

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1468
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

possibly other known identity) against the proxy's identity as presented in the proxy's Certificate message.

The problem is that for a proxy accepted by the client, the proxy needs to be provided information on which grounds it should accept the next-hop certificate. Both the proxy and the user may have rules for this, and the user have the possibility to select the desired behavior. To handle this case, the Accept-Credentials header (See Section 16.2) is used, where the client can force the proxy/proxies to relay back the chain of certificates used to authenticate any intermediate proxies as well as the server. Given the assumption that the proxies are viewed as trusted, it gives the user a possibility to enforce policies to each trusted proxy of whether it should accept the next agent in the chain.

A proxy MUST use TLS for the next hop if the RTSP request includes a "rtsps" URI. TLS MAY be applied on intermediate links (e.g. between client and proxy, or between proxy and proxy), even if the resource and the end server are not require to use it. The proxy MUST, when initiating the next hop TLS connection, use the incoming TLS connections cipher suite list, only modified by removing any cipher suits that the proxy does not support. In case a proxy fails to establish a TLS connection due to cipher suite mismatch between proxy and next hop proxy or server, this is indicated using error code 472 (Failure to establish secure connection).

19.3.1. Accept-Credentials

The Accept-Credentials header can be used by the client to distribute simple authorization policies to intermediate proxies. The client includes the Accept-Credentials header to dictate how the proxy treats the server/next proxy certificate. There are currently three methods defined:

Any, which means that the proxy (or proxies) MUST accept whatever certificate presented. This is of course not a recommended option to use, but may be useful in certain circumstances (such as testing).

Proxy, which means that the proxy (or proxies) MUST use its own policies to validate the certificate and decide whether to accept it or not. This is convenient in cases where the user has a strong trust relation with the proxy. Reason why a strong trust relation may exist are; personal/company proxy, proxy has a out-of-band policy configuration mechanism.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1469
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

User, which means that the proxy (or proxies) MUST send credential information about the next hop to the client for authorization. The client can then decide whether the proxy should accept the certificate or not. See Section 19.3.2 for further details.

If the Accept-Credentials header is not included in the RTSP request from the client, then the "Proxy" method MUST be used as default. If another method than the "Proxy" is to be used, then the Accept-Credentials header MUST be included in all of the RTSP request from the client. This is because it cannot be assumed that the proxy always keeps the TLS state or the users previous preference between different RTSP messages (in particular if the time interval between the messages is long).

With the "Any" and "Proxy" methods the proxy will apply the policy as defined for respectively method. If the policy does not accept the credentials of the next hop, the proxy MUST respond with a message using status code 471 (Connection Credentials not accepted).

An RTSP request in the direction server to client MUST NOT include the Accept-Credential header. As for the non-secured communication, the possibility for these requests depends on the presence of a client established connection. However, if the server to client request is in relation to a session established over a TLS secured channel, it MUST be sent in a TLS secured connection. That secured connection MUST also be the one used by the last client to server request. If no such transport connection exist at the time when the server desires to send the request, the server discard the message.

Further policies MAY be defined and registered, but should be done so with caution.

19.3.2. User approved TLS procedure

For the "User" method, each proxy MUST perform the following procedure for each RTSP request:

- Setup the TLS session to the next hop if not already present (i.e. run the TLS handshake, but do not send the RTSP request).
- Extract the peer certificate chain for the TLS session.
- Check if a matching identity and hash of the peer certificate is present in the Accept-Credentials header. If present, send the message to the next hop, and conclude these procedures. If not, go to the next step.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1470
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- The proxy responds to the RTSP request with a 470 or 407 response code. The 407 response code MAY be used when the proxy requires both user and connection authorization from user or client. In this message the proxy MUST include a Connection-Credentials header, see Section 16.12 with the next hop's identity and certificate.

The client MUST upon receiving a 470 or 407 response with Connection-Credentials header take the decision on whether to accept the certificate or not (if it cannot do so, the user SHOULD be consulted). If the certificate is accepted, the client has to again send the RTSP request. In that request the client has to include the Accept-Credentials header including the hash over the DER encoded certificate for all trusted proxies in the chain.

Example:

```
C->P: SETUP rtsps://test.example.org/secret/audio RTSP/2.0
      CSeq: 2
      Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:4588"/
                 "192.0.2.5:4589"
      Accept-Ranges: NPT, SMPTE, UTC
      Accept-Credentials: User

P->C: RTSP/2.0 470 Connection Authorization Required
      CSeq: 2
      Connection-Credentials: "rtsps://test.example.org";
      MIIDNTCCAp...

C->P: SETUP rtsps://test.example.org/secret/audio RTSP/2.0
      CSeq: 3
      Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:4588"/
                 "192.0.2.5:4589"
      Accept-Credentials: User "rtsps://test.example.org";sha-256;
      dPYD7txpoGTbAqZZQJ+vaeOkyH4=
      Accept-Ranges: NPT, SMPTE, UTC

P->S: SETUP rtsps://test.example.org/secret/audio RTSP/2.0
      CSeq: 3
      Transport: RTP/AVP;unicast;dest_addr="192.0.2.5:4588"/
                 "192.0.2.5:4589"
      Via: RTSP/2.0 proxy.example.org
      Accept-Credentials: User "rtsps://test.example.org";sha-256;
      dPYD7txpoGTbAqZZQJ+vaeOkyH4=
      Accept-Ranges: NPT, SMPTE, UTC
```

One implication of this process is that the connection for secured RTSP messages may take significantly more round-trip times for the

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1471
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

first message. A complete extra message exchange between the proxy connecting to the next hop and the client results because of the process for approval for each hop. However, if each message contains the chain of proxies that the requester accepts, the remaining message exchange should not be delayed. The procedure of including the credentials in each request rather than building state in each proxy, avoids the need for revocation procedures.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1472
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

20. Syntax

The RTSP syntax is described in an Augmented Backus-Naur Form (ABNF) as defined in RFC 5234 [RFC5234]. It uses the basic definitions present in RFC 5234.

Please note that ABNF strings, e.g. "Accept", are case insensitive as specified in section 2.3 of RFC 5234.

20.1. Base Syntax

RTSP header values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream. This is intended to behave exactly as HTTP/1.1 as described in RFC 2616 [RFC2616]. The SWS construct is used when linear white space is optional, generally between tokens and separators.

To separate the header name from the rest of value, a colon is used, which, by the above rule, allows whitespace before, but no line break, and whitespace after, including a line break. The HCOLON defines this construct.

```
OCTET           =  %x00-FF ; any 8-bit sequence of data
CHAR            =  %x01-7F ; any US-ASCII character (octets 1 - 127)
UPALPHA         =  %x41-5A ; any US-ASCII uppercase letter "A".."Z"
LOALPHA         =  %x61-7A ;any US-ASCII lowercase letter "a".."z"
ALPHA           =  UPALPHA / LOALPHA
DIGIT           =  %x30-39 ; any US-ASCII digit "0".."9"
CTL             =  %x00-1F / %x7F  ; any US-ASCII control character
                   ; (octets 0 - 31) and DEL (127)
CR              =  %x0D ; US-ASCII CR, carriage return (13)
LF              =  %x0A  ; US-ASCII LF, linefeed (10)
SP              =  %x20  ; US-ASCII SP, space (32)
HT              =  %x09  ; US-ASCII HT, horizontal-tab (9)
DQ              =  %x22  ; US-ASCII double-quote mark (34)
BACKSLASH       =  %x5C  ; US-ASCII backslash (92)
CRLF            =  CR LF
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1473
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   LWS             =  [CRLF] 1*( SP / HT ) ; Line-breaking White Space
   SWS             =  [LWS] ; Separating White Space
   HCOLON          =  *( SP / HT ) ":" SWS
   TEXT            =  %x20-7E / %x80-FF  ; any OCTET except CTLs
   tspecials       =  "(" / ")" / "<" / ">" / "@"
                   /  "," / ";" / ":" / BACKSLASH / DQ
                   /  "/" / "[" / "]" / "?" / "="
                   /  "{" / "}" / SP / HT
   token           =  1*(%x21 / %x23-27 / %x2A-2B / %x2D-2E / %x30-39
                   /  %x41-5A / %x5E-7A / %x7C / %x7E)
                      ; 1*<any CHAR except CTLs or tspecials>
   quoted-string   =  ( DQ *qdtext DQ )
   qdtext          =  %x20-21 / %x23-7E / %x80-FF / UTF8-NONASCII
                      ; any UTF-8 TEXT except <">
   quoted-pair     =  BACKSLASH CHAR
   ctext           =  %x20-27 / %x2A-7E
                   /  %x80-FF  ; any OCTET except CTLs, "(" and ")"
   generic-param   =  token [ EQUAL gen-value ]
   gen-value       =  token / host / quoted-string

   safe            =  "$" / "-" / "_" / "." / "+"
   extra           =  "!" / "*" / "'" / "(" / ")" / ","
   rtsp-extra      =  "!" / "*" / "'" / "(" / ")"

   HEX             =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
                   /  "a" / "b" / "c" / "d" / "e" / "f"
   LHEX            =  DIGIT /  "a" / "b" / "c" / "d" / "e" / "f"
                      ; lowercase "a-f" Hex
   reserved        =  ";" / "/" / "?" / ":" / "@" / "&" / "="

   unreserved      =  ALPHA / DIGIT / safe / extra
   rtsp-unreserved  =  ALPHA / DIGIT / safe / rtsp-extra

   base64          =  *base64-unit [base64-pad]
   base64-unit     =  4base64-char
   base64-pad      =  (2base64-char "==") / (3base64-char "=")
   base64-char     =  ALPHA / DIGIT / "+" / "/"
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1474
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   SLASH     =  SWS "/" SWS ; slash
   EQUAL     =  SWS "=" SWS ; equal
   LPAREN    =  SWS "(" SWS ; left parenthesis
   RPAREN    =  SWS ")" SWS ; right parenthesis
   COMMA     =  SWS "," SWS ; comma
   SEMI      =  SWS ";" SWS ; semicolon
   COLON     =  SWS ":" SWS ; colon
   MINUS     =  SWS "-" SWS ; minus/dash
   LDQUOT    =  SWS DQ ; open double quotation mark
   RDQUOT    =  DQ SWS ; close double quotation mark
   RAQUOT    =  ">" SWS ; right angle quote
   LAQUOT    =  SWS "<" ; left angle quote

   TEXT-UTF8char     =  %x21-7E / UTF8-NONASCII
   UTF8-NONASCII     =  %xC0-DF 1UTF8-CONT
                     /  %xE0-EF 2UTF8-CONT
                     /  %xF0-F7 3UTF8-CONT
                     /  %xF8-FB 4UTF8-CONT
                     /  %xFC-FD 5UTF8-CONT
   UTF8-CONT         =  %x80-BF

   POS-FLOAT         = 1*12DIGIT ["." 1*9DIGIT]
   FLOAT             = ["-"] POS-FLOAT
```

20.2. RTSP Protocol Definition

20.2.1. Generic Protocol elements

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1475
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   RTSP-IRI       =  schemes ":" IRI-rest
   IRI-rest       =  ihier-part [ "?" iquery ] [ "#" ifragment ]
   ihier-part     =  "//" iauthority ipath-abempty
   RTSP-IRI-ref   =  RTSP-IRI / irelative-ref
   irelative-ref  =  irelative-part [ "?" iquery ] [ "#" ifragment ]
   irelative-part =  "//" iauthority ipath-abempty
                     / ipath-absolute
                     / ipath-noscheme
                     / ipath-empty

   iauthority     =  < As defined in RFC 3987>
   ipath          =  ipath-abempty   ; begins with "/" or is empty
                     / ipath-absolute  ; begins with "/" but not "//"
                     / ipath-noscheme  ; begins with a non-colon segment
                     / ipath-rootless  ; begins with a segment
                     / ipath-empty     ; zero characters

   ipath-abempty   =  *( "/" isegment )
   ipath-absolute  =  "/" [ isegment-nz *( "/" isegment ) ]
   ipath-noscheme  =  isegment-nz-nc *( "/" isegment )
   ipath-rootless  =  isegment-nz *( "/" isegment )
   ipath-empty     =  0<ipchar>

   isegment        =  *ipchar [";" *ipchar]
   isegment-nz     =  1*ipchar [";" *ipchar]
                      / ";" *ipchar
   isegment-nz-nc  =  (1*ipchar-nc [";" *ipchar-nc])
                      / ";" *ipchar-nc
                      ; non-zero-length segment without any colon ":"

   ipchar          =  iunreserved / pct-encoded / sub-delims / ":" / "@"
   ipchar-nc       =  iunreserved / pct-encoded / sub-delims / "@"

   iquery          =  < As defined in RFC 3987>
   ifragment       =  < As defined in RFC 3987>
   iunreserved     =  < As defined in RFC 3987>
   pct-encoded     =  < As defined in RFC 3987>
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1476
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   RTSP-URI        =  schemes ":" URI-rest
   RTSP-REQ-URI    =  schemes ":" URI-req-rest
   RTSP-URI-Ref    =  RTSP-URI / RTSP-Relative
   RTSP-REQ-Ref    =  RTSP-REQ-URI / RTSP-REQ-Rel
   schemes         =  "rtsp" / "rtsps" / scheme
   scheme          =  < As defined in RFC 3986>
   URI-rest        =  hier-part [ "?" query ] [ "#" fragment ]
   URI-req-rest    =  hier-part [ "?" query ]
                      ; Note fragment part not allowed in requests
   hier-part       =  "//" authority path-abempty

   RTSP-Relative   =  relative-part [ "?" query ] [ "#" fragment ]
   RTSP-REQ-Rel    =  relative-part [ "?" query ]
   relative-part   =  "//" authority path-abempty
                      / path-absolute
                      / path-noscheme
                      / path-empty

   authority       =  < As defined in RFC 3986>
   query           =  < As defined in RFC 3986>
   fragment        =  < As defined in RFC 3986>

   path            =  path-abempty    ; begins with "/" or is empty
                      / path-absolute ; begins with "/" but not "//"
                      / path-noscheme ; begins with a non-colon segment
                      / path-rootless ; begins with a segment
                      / path-empty    ; zero characters

   path-abempty    =  *( "/" segment )
   path-absolute   =  "/" [ segment-nz *( "/" segment ) ]
   path-noscheme   =  segment-nz-nc *( "/" segment )
   path-rootless   =  segment-nz *( "/" segment )
   path-empty      =  0<pchar>

   segment         =  *pchar [";" *pchar]
   segment-nz      =  ( 1*pchar [";" *pchar]) / (";" *pchar)
   segment-nz-nc   =  ( 1*pchar-nc [";" *pchar-nc]) / (";" *pchar-nc)
                      ; non-zero-length segment without any colon ":"

   pchar           =  unreserved / pct-encoded / sub-delims / ":" / "@"
   pchar-nc        =  unreserved / pct-encoded / sub-delims / "@"

   sub-delims      =  "!" / "$" / "&" / "'" / "(" / ")"
                      / "*" / "+" / "," / "="
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1477
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   smpte-range        =  smpte-type ["=" smpte-range-spec]
                         ; See section 3.4
   smpte-range-spec   =  ( smpte-time "-" [ smpte-time ] )
                      /  ( "-" smpte-time )
   smpte-type         =  "smpte" / "smpte-30-drop"
                      /  "smpte-25" / smpte-type-extension
                      ; other timecodes may be added
   smpte-type-extension  =  "smpte" token
   smpte-time         =  1*2DIGIT ":" 1*2DIGIT ":" 1*2DIGIT
                         [ ":" 1*2DIGIT [ "." 1*2DIGIT ] ]


   npt-range        =  "npt" ["=" npt-range-spec]
   npt-range-spec   =  ( npt-time "-" [ npt-time ] ) / ( "-" npt-time )
   npt-time         =  "now" / npt-sec / npt-hhmmss
   npt-sec          =  1*19DIGIT [ "." 1*9DIGIT ]
   npt-hhmmss       =  npt-hh ":" npt-mm ":" npt-ss [ "." 1*9DIGIT ]
   npt-hh           =  1*19DIGIT   ; any positive number
   npt-mm           =  1*2DIGIT  ; 0-59
   npt-ss           =  1*2DIGIT  ; 0-59


   utc-range        =  "clock" ["=" utc-range-spec]
   utc-range-spec   =  ( utc-time "-" [ utc-time ] ) / ( "-" utc-time )
   utc-time         =  utc-date "T" utc-clock "Z"
   utc-date         =  8DIGIT
   utc-clock        =  6DIGIT [ "." 1*9DIGIT ]


   feature-tag       =  token

   session-id        =  1*256( ALPHA / DIGIT / safe )

   extension-header  =  header-name HCOLON header-value
   header-name       =  token
   header-value      =  *(TEXT-UTF8char / UTF8-CONT / LWS)
```

20.2.2. Message Syntax

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1478
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   RTSP-message  = Request / Response  ; RTSP/2.0 messages

   Request       = Request-Line
                   *((general-header
                   /  request-header
                   /  message-header) CRLF)
                   CRLF
                   [ message-body-data ]

   Response      = Status-Line
                   *((general-header
                   /  response-header
                   /  message-header) CRLF)
                   CRLF
                   [ message-body-data ]

   Request-Line = Method SP Request-URI SP RTSP-Version CRLF

   Status-Line  = RTSP-Version SP Status-Code SP Reason-Phrase CRLF
   Method  =  "DESCRIBE"
           /  "GET_PARAMETER"
           /  "OPTIONS"
           /  "PAUSE"
           /  "PLAY"
           /  "PLAY_NOTIFY"
           /  "REDIRECT"
           /  "SETUP"
           /  "SET_PARAMETER"
           /  "TEARDOWN"
           /  extension-method

   extension-method  =  token

   Request-URI  =  "*" / RTSP-REQ-URI
   RTSP-Version =  "RTSP/" 1*DIGIT "." 1*DIGIT

   message-body-data = 1*OCTET

   Status-Code  =  "100"  ; Continue
                /  "200"  ; OK
                /  "301"  ; Moved Permanently
                /  "302"  ; Found
                /  "303"  ; See Other
                /  "304"  ; Not Modified
                /  "305"  ; Use Proxy
                /  "400"  ; Bad Request
                /  "401"  ; Unauthorized
                /  "402"  ; Payment Required
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1479
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
                /  "403"  ; Forbidden
                /  "404"  ; Not Found
                /  "405"  ; Method Not Allowed
                /  "406"  ; Not Acceptable
                /  "407"  ; Proxy Authentication Required
                /  "408"  ; Request Time-out
                /  "410"  ; Gone
                /  "411"  ; Length Required
                /  "412"  ; Precondition Failed
                /  "413"  ; Request Message Body Too Large
                /  "414"  ; Request-URI Too Large
                /  "415"  ; Unsupported Media Type
                /  "451"  ; Parameter Not Understood
                /  "452"  ; reserved
                /  "453"  ; Not Enough Bandwidth
                /  "454"  ; Session Not Found
                /  "455"  ; Method Not Valid in This State
                /  "456"  ; Header Field Not Valid for Resource
                /  "457"  ; Invalid Range
                /  "458"  ; Parameter Is Read-Only
                /  "459"  ; Aggregate operation not allowed
                /  "460"  ; Only aggregate operation allowed
                /  "461"  ; Unsupported Transport
                /  "462"  ; Destination Unreachable
                /  "463"  ; Destination Prohibited
                /  "464"  ; Data Transport Not Ready Yet
                /  "465"  ; Notification Reason Unknown
                /  "466"  ; Key Management Error
                /  "470"  ; Connection Authorization Required
                /  "471"  ; Connection Credentials not accepted
                /  "472"  ; Failure to establish secure connection
                /  "500"  ; Internal Server Error
                /  "501"  ; Not Implemented
                /  "502"  ; Bad Gateway
                /  "503"  ; Service Unavailable
                /  "504"  ; Gateway Time-out
                /  "505"  ; RTSP Version not supported
                /  "551"  ; Option not supported
                /  extension-code

   extension-code  =  3DIGIT

   Reason-Phrase   =  1*(TEXT-UTF8char / HT / SP)
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1480
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   general-header  =  Cache-Control
                   /  Connection
                   /  CSeq
                   /  Date
                   /  Media-Properties
                   /  Media-Range
                   /  Pipelined-Requests
                   /  Proxy-Supported
                   /  Seek-Style
                   /  Server
                   /  Supported
                   /  Timestamp
                   /  User-Agent
                   /  Via
                   /  extension-header

   request-header  =  Accept
                   /  Accept-Credentials
                   /  Accept-Encoding
                   /  Accept-Language
                   /  Authorization
                   /  Bandwidth
                   /  Blocksize
                   /  From
                   /  If-Match
                   /  If-Modified-Since
                   /  If-None-Match
                   /  Notify-Reason
                   /  Proxy-Require
                   /  Range
                   /  Referrer
                   /  Request-Status
                   /  Require
                   /  Scale
                   /  Session
                   /  Speed
                   /  Supported
                   /  Terminate-Reason
                   /  Transport
                   /  extension-header
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1481
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
response-header  =  Accept-Credentials
                 /  Accept-Ranges
                 /  Connection-Credentials
                 /  MTag
                 /  Location
                 /  Proxy-Authenticate
                 /  Public
                 /  Range
                 /  Retry-After
                 /  RTP-Info
                 /  Scale
                 /  Session
                 /  Speed
                 /  Transport
                 /  Unsupported
                 /  Vary
                 /  WWW-Authenticate
                 /  extension-header

message-header    =  Allow
                 /  Content-Base
                 /  Content-Encoding
                 /  Content-Language
                 /  Content-Length
                 /  Content-Location
                 /  Content-Type
                 /  Expires
                 /  Last-Modified
                 /  extension-header
```

20.2.3. Header Syntax

```
Accept            =  "Accept" HCOLON
                     [ accept-range *(COMMA accept-range) ]
accept-range      =  media-type-range [SEMI accept-params]
media-type-range  =  ( "*/*"
                     / ( m-type SLASH "*" )
                     / ( m-type SLASH m-subtype )
                    ) *( SEMI m-parameter )
accept-params     =  "q" EQUAL qvalue *(SEMI generic-param )
qvalue            =  ( "0" [ "." *3DIGIT ] )
                  /  ( "1" [ "." *3("0") ] )
Accept-Credentials =  "Accept-Credentials" HCOLON cred-decision
cred-decision     =  ("User" [LWS cred-info])
                  /  "Proxy"
                  /  "Any"
                  /  (token [LWS 1*header-value])
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1482
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
                                    ; For future extensions
   cred-info          =  cred-info-data *(COMMA cred-info-data)

   cred-info-data     =  DQ RTSP-REQ-URI DQ SEMI hash-alg SEMI base64
   hash-alg           =  "sha-256" / extension-alg
   extension-alg      =  token
   Accept-Encoding    =  "Accept-Encoding" HCOLON
                         [ encoding *(COMMA encoding) ]
   encoding           =  codings [SEMI accept-params]
   codings            =  content-coding / "*"
   content-coding     =  token
   Accept-Language    =  "Accept-Language" HCOLON
                         language *(COMMA language)
   language           =  language-range [SEMI accept-params]
   language-range     =  language-tag / "*"
   language-tag       =  primary-tag *( "-" subtag )
   primary-tag        =  1*8ALPHA
   subtag             =  1*8ALPHA
   Accept-Ranges      =  "Accept-Ranges" HCOLON acceptable-ranges
   acceptable-ranges  =  (range-unit *(COMMA range-unit))
   range-unit         =  "NPT" / "SMPTE" / "UTC" / extension-format
   extension-format   =  token
   Allow              =  "Allow" HCOLON Method *(COMMA Method)
   Authorization      =  "Authorization" HCOLON credentials
   credentials        =  ("Digest" LWS digest-response)
                      /  other-response
   digest-response    =  dig-resp *(COMMA dig-resp)
   dig-resp           =  username / realm / nonce / digest-uri
                      /  dresponse / algorithm / cnonce
                      /  opaque / message-qop
                      /  nonce-count / auth-param
   username           =  "username" EQUAL username-value
   username-value     =  quoted-string
   digest-uri         =  "uri" EQUAL LDQUOT digest-uri-value RDQUOT
   digest-uri-value   =  RTSP-REQ-URI
   message-qop        =  "qop" EQUAL qop-value
   cnonce             =  "cnonce" EQUAL cnonce-value
   cnonce-value       =  nonce-value
   nonce-count        =  "nc" EQUAL nc-value
   nc-value           =  8LHEX
   dresponse          =  "response" EQUAL request-digest
   request-digest     =  LDQUOT 32LHEX RDQUOT
   auth-param         =  auth-param-name EQUAL
                         ( token / quoted-string )
   auth-param-name    =  token
   other-response     =  auth-scheme LWS auth-param
                         *(COMMA auth-param)
   auth-scheme        =  token
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1483
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   Bandwidth          =  "Bandwidth" HCOLON 1*19DIGIT

   Blocksize          =  "Blocksize" HCOLON 1*9DIGIT

   Cache-Control      =  "Cache-Control" HCOLON cache-directive
                         *(COMMA cache-directive)
   cache-directive    =  cache-rqst-directive
                      /  cache-rspns-directive

   cache-rqst-directive =  "no-cache"
                        /  "max-stale" [EQUAL delta-seconds]
                        /  "min-fresh" EQUAL delta-seconds
                        /  "only-if-cached"
                        /  cache-extension

   cache-rspns-directive =  "public"
                         /  "private"
                         /  "no-cache"
                         /  "no-transform"
                         /  "must-revalidate"
                         /  "proxy-revalidate"
                         /  "max-age" EQUAL delta-seconds
                         /  cache-extension

   cache-extension    =  token [EQUAL (token / quoted-string)]
   delta-seconds      =  1*19DIGIT

   Connection          =  "Connection" HCOLON connection-token
                          *(COMMA connection-token)
   connection-token    =  "close" / token

   Connection-Credentials = "Connection-Credentials" HCOLON cred-chain
   cred-chain          =  DQ RTSP-REQ-URI DQ SEMI base64

   Content-Base        =  "Content-Base" HCOLON RTSP-URI
   Content-Encoding    =  "Content-Encoding" HCOLON
                          content-coding *(COMMA content-coding)
   Content-Language    =  "Content-Language" HCOLON
                          language-tag *(COMMA language-tag)
   Content-Length      =  "Content-Length" HCOLON 1*19DIGIT
   Content-Location    =  "Content-Location" HCOLON RTSP-REQ-Ref
   Content-Type        =  "Content-Type" HCOLON media-type
   media-type          =  m-type SLASH m-subtype *(SEMI m-parameter)
   m-type              =  discrete-type / composite-type
   discrete-type       =  "text" / "image" / "audio" / "video"
                       /  "application" / extension-token
   composite-type    =  "message" / "multipart" / extension-token
   extension-token   =  ietf-token / x-token
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1484
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   ietf-token        =  token
   x-token           =  "x-" token
   m-subtype         =  extension-token / iana-token
   iana-token        =  token
   m-parameter       =  m-attribute EQUAL m-value
   m-attribute       =  token
   m-value           =  token / quoted-string

   CSeq              =  "CSeq" HCOLON cseq-nr
   cseq-nr           =  1*9DIGIT
   Date              =  "Date" HCOLON RTSP-date
   RTSP-date         =  rfc1123-date ; HTTP-date
   rfc1123-date      =  wkday "," SP date1 SP time SP "GMT"
   date1             =  2DIGIT SP month SP 4DIGIT
                        ; day month year (e.g., 02 Jun 1982)
   time              =  2DIGIT ":" 2DIGIT ":" 2DIGIT
                        ; 00:00:00 - 23:59:59
   wkday             =  "Mon" / "Tue" / "Wed"
                     /  "Thu" / "Fri" / "Sat" / "Sun"
   month             =  "Jan" / "Feb" / "Mar" / "Apr"
                     /  "May" / "Jun" / "Jul" / "Aug"
                     /  "Sep" / "Oct" / "Nov" / "Dec"

   Expires           =  "Expires" HCOLON RTSP-date
   From              =  "From" HCOLON from-spec
   from-spec         =  ( name-addr / addr-spec ) *( SEMI from-param )
   name-addr         =  [ display-name ] LAQUOT addr-spec RAQUOT
   addr-spec         =  RTSP-REQ-URI / absolute-URI
   absolute-URI      =  < As defined in RFC 3986>
   display-name      =  *(token LWS) / quoted-string
   from-param        =  tag-param / generic-param
   tag-param         =  "tag" EQUAL token
   If-Match          =  "If-Match" HCOLON ("*" / message-tag-list)
   message-tag-list  =  message-tag *(COMMA message-tag)
   message-tag       =  [ weak ] opaque-tag
   weak              =  "W/"
   opaque-tag        =  quoted-string
   If-Modified-Since  =  "If-Modified-Since" HCOLON RTSP-date
   If-None-Match     =  "If-None-Match" HCOLON ("*" / message-tag-list)
   Last-Modified     =  "Last-Modified" HCOLON RTSP-date
   Location          =  "Location" HCOLON RTSP-REQ-URI
   Media-Properties  = "Media-Properties" HCOLON [media-prop-list]
   media-prop-list   = media-prop-value *(COMMA media-prop-value)
   media-prop-value  = ("Random-Access" [EQUAL POS-FLOAT])
                     / "Begining-Only"
                     / "No-Seeking"
                     / "Immutable"
                     / "Dynamic"
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1485
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
                    / "Time-Progressing"
                    / "Unlimited"
                    / ("Time-Limited" EQUAL utc-time)
                    / ("Time-Duration" EQUAL POS-FLOAT)
                    / ("Scales" EQUAL scale-value-list)
                    / media-prop-ext
   media-prop-ext   = token [EQUAL (1*rtsp-unreserved / quoted-string)]
   scale-value-list = DQ scale-entry *(COMMA scale-entry) DQ
   scale-entry      = scale-value / (scale-value COLON scale-value)
   scale-value      = FLOAT
   Media-Range      = "Media-Range" HCOLON [ranges-list]
   ranges-list      =  ranges-spec *(COMMA ranges-spec)
   MTag             =  "MTag" HCOLON message-tag
   Notify-Reason    = "Notify-Reason" HCOLON Notify-Reas-val
   Notify-Reas-val  = "end-of-stream"
                    / "media-properties-update"
                    / "scale-change"
                    / Notify-Reason-extension
   Notify-Reason-extension  = token
   Pipelined-Requests = "Pipelined-Requests" HCOLON startup-id
   startup-id  = 1*8DIGIT
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1486
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
Proxy-Authenticate   =  "Proxy-Authenticate" HCOLON challenge-list
challenge-list       =  challenge *(COMMA challenge)
challenge            =  ("Digest" LWS digest-cln *(COMMA digest-cln))
                     /  other-challenge
other-challenge      =  auth-scheme LWS auth-param
                        *(COMMA auth-param)
digest-cln           =  realm / domain / nonce
                     /  opaque / stale / algorithm
                     /  qop-options / auth-param
realm                =  "realm" EQUAL realm-value
realm-value          =  quoted-string
domain               =  "domain" EQUAL LDQUOT RTSP-REQ-Ref
                        *(1*SP RTSP-REQ-Ref ) RDQUOT
nonce                =  "nonce" EQUAL nonce-value
nonce-value          =  quoted-string
opaque               =  "opaque" EQUAL quoted-string
stale                =  "stale" EQUAL ( "true" / "false" )
algorithm            =  "algorithm" EQUAL ("MD5" / "MD5-sess" / token)
qop-options          =  "qop" EQUAL LDQUOT qop-value
                        *("," qop-value) RDQUOT
qop-value            =  "auth" / "auth-int" / token
Proxy-Require        =  "Proxy-Require" HCOLON feature-tag-list
feature-tag-list     =  feature-tag *(COMMA feature-tag)
Proxy-Supported      =  "Proxy-Supported" HCOLON [feature-tag-list]

Public               =  "Public" HCOLON Method *(COMMA Method)

Range                =  "Range" HCOLON ranges-spec

ranges-spec          =  npt-range / utc-range / smpte-range
                     /  range-ext
range-ext            =  extension-format ["=" range-value]
range-value          =  1*(rtsp-unreserved / quoted-string / ":" )

Referrer             =  "Referrer" HCOLON (absolute-URI / RTSP-URI-Ref)
Request-Status       =  "Request-Status" HCOLON req-status-info
req-status-info      =  cseq-info LWS status-info LWS reason-info
cseq-info            =  "cseq" EQUAL cseq-nr
status-info          =  "status" EQUAL Status-Code
reason-info          =  "reason" EQUAL DQ Reason-Phrase DQ
Require              =  "Require" HCOLON feature-tag-list
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1487
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
RTP-Info          =  "RTP-Info" HCOLON [rtsp-info-spec
                     *(COMMA rtsp-info-spec)]
rtsp-info-spec    =  stream-url 1*ssrc-parameter
stream-url        =  "url" EQUAL DQ RTSP-REQ-Ref DQ
ssrc-parameter    =  LWS "ssrc" EQUAL ssrc HCOLON
                     ri-parameter *(SEMI ri-parameter)
ri-parameter      =  ("seq" EQUAL 1*5DIGIT)
                  /  ("rtptime" EQUAL 1*10DIGIT)
                  /  generic-param

Retry-After       =  "Retry-After" HCOLON ( RTSP-date / delta-seconds )
Scale             =  "Scale" HCOLON scale-value
Seek-Style        =  "Seek-Style" HCOLON Seek-S-values
Seek-S-values     =  "RAP"
                  /  "CoRAP"
                  /  "First-Prior"
                  /  "Next"
                  /  Seek-S-value-ext
Seek-S-value-ext  =  token

Server            =  "Server" HCOLON ( product / comment )
                     *(LWS (product / comment))
product           =  token [SLASH product-version]
product-version   =  token
comment           =  LPAREN *( ctext / quoted-pair) RPAREN

Session           =  "Session" HCOLON session-id
                     [ SEMI "timeout" EQUAL delta-seconds ]

Speed             =  "Speed" HCOLON lower-bound MINUS upper-bound
lower-bound       =  POS-FLOAT
upper-bound       =  POS-FLOAT

Supported         =  "Supported" HCOLON [feature-tag-list]
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1488
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   Terminate-Reason      =  "Terminate-Reason" HCOLON TR-Info
   TR-Info               =  TR-Reason *(SEMI TR-Parameter)
   TR-Reason             =  "Session-Timeout"
                         /  "Server-Admin"
                         /  "Internal-Error"
                         /  token
   TR-Parameter          =  TR-time / TR-user-msg / generic-param
   TR-time               =  "time" EQUAL utc-time
   TR-user-msg           =  "user-msg" EQUAL quoted-string

   Timestamp        =  "Timestamp" HCOLON timestamp-value [LWS delay]
   timestamp-value  =  *19DIGIT [ "." *9DIGIT ]
   delay            =  *9DIGIT [ "." *9DIGIT ]

   Transport        =  "Transport" HCOLON transport-spec
                       *(COMMA transport-spec)
   transport-spec   =  transport-id *trns-parameter
   transport-id     =  trans-id-rtp / other-trans
   trans-id-rtp     =  "RTP/" profile ["/" lower-transport]
                       ; no LWS is allowed inside transport-id
   other-trans      =  token *("/" token)
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1489
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
profile              = "AVP" / "SAVP" / "AVPF" / token
lower-transport      = "TCP" / "UDP" / token
trns-parameter       = (SEMI ( "unicast" / "multicast" ))
                     / (SEMI "interleaved" EQUAL channel [ "-" channel ])
                     / (SEMI "ttl" EQUAL ttl)
                     / (SEMI "layers" EQUAL 1*DIGIT)
                     / (SEMI "ssrc" EQUAL ssrc *(SLASH ssrc))
                     / (SEMI "mode" EQUAL mode-spec)
                     / (SEMI "dest_addr" EQUAL addr-list)
                     / (SEMI "src_addr" EQUAL addr-list)
                     / (SEMI "setup" EQUAL contrans-setup)
                     / (SEMI "connection" EQUAL contrans-con)
                     / (SEMI "RTCP-mux")
                     / (SEMI "MIKEY" EQUAL MIKEY-Value)
                     / (SEMI trn-param-ext)
contrans-setup       = "active" / "passive" / "actpass"
contrans-con         = "new" / "existing"
trn-param-ext        = par-name [EQUAL trn-par-value]
par-name             = token
trn-par-value        = *(rtsp-unreserved / quoted-string)
ttl                  = 1*3DIGIT ; 0 to 255
ssrc                 = 8HEX
channel              = 1*3DIGIT ; 0 to 255
MIKEY-Value          = base64
mode-spec            = ( DQ mode *(COMMA mode) DQ )
mode                 = "PLAY" / token
addr-list            = quoted-addr *(SLASH quoted-addr)
quoted-addr          = DQ (host-port / extension-addr) DQ
host-port            = ( host [":" port] )
                     / ( ":" port )
extension-addr       = 1*qdtext
host                 = < As defined in RFC 3986>
port                 = < As defined in RFC 3986>
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1490
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

```
   Unsupported     = "Unsupported" HCOLON feature-tag-list

   User-Agent      = "User-Agent" HCOLON ( product / comment )
                     *(LWS (product / comment))

   Vary            = "Vary" HCOLON ( "*" / field-name-list)
   field-name-list = field-name *(COMMA field-name)
   field-name      = token
   Via             = "Via" HCOLON via-parm *(COMMA via-parm)
   via-parm        = sent-protocol LWS sent-by *( SEMI via-params )
   via-params      = via-ttl / via-maddr
                   / via-received / via-extension
   via-ttl         = "ttl" EQUAL ttl
   via-maddr       = "maddr" EQUAL host
   via-received    = "received" EQUAL (IPv4address / IPv6address)
   IPv4address     = < As defined in RFC 3986>
   IPv6address     = < As defined in RFC 3986>
   via-extension   = generic-param
   sent-protocol   = protocol-name SLASH protocol-version
                     SLASH transport-prot
   protocol-name   = "RTSP" / token
   protocol-version = token
   transport-prot  = "UDP" / "TCP" / "TLS" / other-transport
   other-transport = token
   sent-by         = host [ COLON port ]

   WWW-Authenticate = "WWW-Authenticate" HCOLON challenge-list
```

20.3. SDP extension Syntax

This section defines in ABNF the SDP extensions defined for RTSP. See Appendix D for the definition of the extensions in text.

```
   control-attribute   =  "a=control:" *SP RTSP-REQ-Ref CRLF

   a-range-def         =  "a=range:" ranges-spec CRLF

   a-mtag-def          =  "a=mtag:" message-tag CRLF
```


Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1491
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

21. Security Considerations

Because of the similarity in syntax and usage between RTSP servers and HTTP servers, the security considerations outlined in [H15] apply also.

Specifically, please note the following:

Abuse of Server Log Information: RTSP and HTTP servers will presumably have similar logging mechanisms, and thus should be equally guarded in protecting the contents of those logs, thus protecting the privacy of the users of the servers. See [H15.1.1] for HTTP server recommendations regarding server logs.

Transfer of Sensitive Information: There is no reason to believe that information transferred or controlled via RTSP may be any less sensitive than that normally transmitted via HTTP. Therefore, all of the precautions regarding the protection of data privacy and user privacy apply to implementors of RTSP clients, servers, and proxies. See [H15.1.2] for further details.

Attacks Based On File and Path Names: Though RTSP URIs are opaque handles that do not necessarily have file system semantics, it is anticipated that many implementations will translate portions of the Request-URIs directly to file system calls. In such cases, file systems SHOULD follow the precautions outlined in [H15.5], such as checking for ".." in path components.

Personal Information: RTSP clients are often privy to the same information that HTTP clients are (user name, location, etc.) and thus should be equally sensitive. See [H15.1] for further recommendations.

Privacy Issues Connected to Accept Headers: Since may of the same "Accept" headers exist in RTSP as in HTTP, the same caveats outlined in [H15.1.4] with regards to their use should be followed.

DNS Spoofing: Presumably, given the longer connection times typically associated to RTSP sessions relative to HTTP sessions, RTSP client DNS optimizations should be less prevalent. Nonetheless, the recommendations provided in [H15.3] are still relevant to any implementation which attempts to rely on a DNS-to-IP mapping to hold beyond a single use of the mapping.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1492
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Location Headers and Spoofing: If a single server supports multiple organizations that do not trust each another, then it needs to check the values of Location and Content-Location header fields in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority. ([H15.4])

In addition to the recommendations in the current HTTP specification (RFC 2616 [RFC2616], as of this writing) and also of the previous RFC 2068 [RFC2068], future HTTP specifications may provide additional guidance on security issues.

The following are added considerations for RTSP implementations.

Concentrated denial-of-service attack: The protocol offers the opportunity for a remote-controlled denial-of-service attack. See Section 21.1.

Session hijacking: Since there is no or little relation between a transport layer connection and an RTSP session, it is possible for a malicious client to issue requests with random session identifiers which would affect unsuspecting clients. The server SHOULD use a large, random and non-sequential session identifier to minimize the possibility of this kind of attack. However, unless the RTSP signaling always are confidentiality protected, e.g. using TLS, an on-path attacker will be able to hijack a session. For real session security, client authentication needs to be performed.

Authentication: Servers SHOULD implement both basic and digest [RFC2617] authentication. In environments requiring tighter security for the control messages, the transport layer mechanism TLS [RFC5246] SHOULD be used.

Stream issues: RTSP only provides for stream control. Stream delivery issues are not covered in this section, nor in the rest of this draft. RTSP implementations will most likely rely on other protocols such as RTP, IP multicast, RSVP and IGMP, and should address security considerations brought up in those and other applicable specifications.

Persistently suspicious behavior: RTSP servers SHOULD return error code 403 (Forbidden) upon receiving a single instance of behavior which is deemed a security risk. RTSP servers SHOULD also be aware of attempts to probe the server for weaknesses and entry points and MAY arbitrarily disconnect and ignore further requests clients which are deemed to be in violation of

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1493
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

local security policy.

Scope of Multicast: If RTSP is used to control the transmission of media onto a multicast network it is need to consider the scope that delivery has. RTSP supports the TTL Transport header parameter to indicate this scope. However, such scope control is risk as it may be set to large and distribute media beyond the intended scope.

TLS through proxies: If one uses the possibility to connect TLS in multiple legs (Section 19.3 one really needs to be aware of the trust model. That procedure requires full faith and trust in all proxies that one allows to connect through. They are man in the middle and has access to all that goes on over the TLS connection. Thus it is important to consider if that trust model is acceptable in the actual application.

Resource Exhaustion: As RTSP is a stateful protocol and establish resource usages on the server there is a clear possibility to attack the server by trying to overbook these resources to perform an denial of service attack. This attack can be both against ongoing sessions and to prevent others from establishing sessions. RTSP agents will need to have mechanism to prevent single peers from consuming extensive amounts of resources.

21.1. Remote denial of Service Attack

The attacker may initiate traffic flows to one or more IP addresses by specifying them as the destination in SETUP requests. While the attacker's IP address may be known in this case, this is not always useful in prevention of more attacks or ascertaining the attackers identity. Thus, an RTSP server MUST only allow client-specified destinations for RTSP-initiated traffic flows if the server has ensured that the specified destination address accepts receiving media through different security mechanisms. Security mechanisms that are acceptable in an increased generality are:

- Verification of the client's identity, either against a database of known users using RTSP authentication mechanisms (preferably digest authentication or stronger)
- A list of addresses that accept to be media destinations, especially considering user identity
- Media path based verification

The server SHOULD NOT allow the destination field to be set unless a

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1494
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

mechanism exists in the system to authorize the request originator to direct streams to the recipient. It is preferred that this authorization be performed by the media recipient (destination) itself and the credentials passed along to the server. However, in certain cases, such as when recipient address is a multicast group, or when the recipient is unable to communicate with the server in an out-of-band manner, this may not be possible. In these cases the server may chose another method such as a server-resident authorization list to ensure that the request originator has the proper credentials to request stream delivery to the recipient.

One solution that performs the necessary verification of acceptance of media suitable for unicast based delivery is the ICE based NAT traversal method described in [I-D.ietf-mmusic-rtsp-nat]. By using random passwords and username the probability of unintended indication as a valid media destination is very low. If the server include in its STUN requests a cookie (consisting of random material) that is the destination echo back the solution is also safe against having a off-path attacker being able to spoof the STUN checks. Leaving this solution vulnerable only to on-path attackers that can see the STUN requests go to the target of attack.

For delivery to multicast addresses there is need for another solution which is not specified here.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1495
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22. IANA Considerations

This section sets up a number of registries for RTSP 2.0 that should be maintained by IANA. These registries are separate from any registries existing for RTSP 1.0. For each registry there is a description on what it is required to contain, what specification is needed when adding a entry with IANA, and finally the entries that this document needs to register. See also the Section 2.7 "Extending RTSP". There is also an IANA registration of two SDP attributes.

The sections describing how to register an item uses some of the requirements level described in RFC 5226 [RFC5226], namely "First Come, First Served", "Expert Review, "Specification Required", and "Standards Action".

In case a registry requires a contact person, the authors are the contact person for any entries created by this document.

A registration request to IANA MUST contain the following information:

- A name of the item to register according to the rules specified by the intended registry.
- Indication of who has change control over the feature (for example, IETF, ISO, ITU-T, other international standardization bodies, a consortium, a particular company or group of companies, or an individual);
- A reference to a further description, if available, for example (in decreasing order of preference) an RFC, a published standard, a published paper, a patent filing, a technical report, documented source code or a computer manual;
- For proprietary features, contact information (postal and email address);

22.1. Feature-tags

22.1.1. Description

When a client and server try to determine what part and functionality of the RTSP specification and any future extensions that its counter part implements there is need for a namespace. This registry contains named entries representing certain functionality.

The usage of feature-tags is explained in Section 11 and Section 13.1.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1496
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.1.2. Registering New Feature-tags with IANA

The registering of feature-tags is done on a first come, first served basis.

The name of the feature MUST follow these rules: The name may be of any length, but SHOULD be no more than twenty characters long. The name MUST NOT contain any spaces, or control characters. The registration MUST indicate if the feature-tag applies to clients, servers, or proxies only or any combinations of these. Any proprietary feature MUST have as the first part of the name a vendor tag, which identifies the organization. The registry entries consists of the tag, a one paragraph description of what it represents, its applicability (server, client, proxy, any combination) and a reference to its specification where applicable.

22.1.3. Registered entries

The following feature-tags are in this specification defined and hereby registered. The change control belongs to the IETF.

play.basic: The minimal implementation for delivery and playback operations according to this specification. Applies for both clients, servers and proxies.

play.scale: Support of scale operations for media playback. Applies only for servers.

play.speed: Support of the speed functionality for media delivery. Applies only for servers.

setup.rtp.rtcp.mux Support of the RTP and RTCP multiplexing as discussed in Appendix C.1.6.4. Applies for both client and servers and any media caching proxy.

This should be represented by IANA as table with the feature tags, contact person and their references.

22.2. RTSP Methods

22.2.1. Description

What a method is, is described in section Section 13. Extending the protocol with new methods allow for totally new functionality.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1497
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.2.2. Registering New Methods with IANA

A new method MUST be registered through an IETF Standards Action. The reason is that new methods may radically change the protocol's behavior and purpose.

A specification for a new RTSP method MUST consist of the following items:

- A method name which follows the ABNF rules for methods.
- A clear specification what a request using the method does and what responses are expected. Which directions the method is used, C->S or S->C or both. How the use of headers, if any, modifies the behavior and effect of the method.
- A list or table specifying which of the registered headers that are allowed to use with the method in request or/and response.
- Describe how the method relates to network proxies.

22.2.3. Registered Entries

This specification, RFCXXXX, registers 10 methods: DESCRIBE, GET_PARAMETER, OPTIONS, PAUSE, PLAY, PLAY_NOTIFY REDIRECT, SETUP, SET_PARAMETER, and TEARDOWN. The initial table of the registry is below provided.

Method	Directionality	Reference
DESCRIBE	C->S	[RFCXXXX]
GET_PARAMETER	C->S, S->C	[RFCXXXX]
OPTIONS	C->S, S->C	[RFCXXXX]
PAUSE	C->S	[RFCXXXX]
PLAY	C->S	[RFCXXXX]
PLAY_NOTIFY	S->C	[RFCXXXX]
REDIRECT	S->C	[RFCXXXX]
SETUP	C->S	[RFCXXXX]
SET_PARAMETER	C->S, S->C	[RFCXXXX]
TEARDOWN	C->S, S->C	[RFCXXXX]

22.3. RTSP Status Codes

22.3.1. Description

A status code is the three digit numbers used to convey information in RTSP response messages, seeSection 8. The number space is limited and care should be taken not to fill the space.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1498
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.3.2. Registering New Status Codes with IANA

A new status code registrations follows the policy of IETF Review. A specification for a new status code MUST specify the following:

- The requested number.
- A description what the status code means and the expected behavior of the sender and receiver of the code.

22.3.3. Registered Entries

RFCXXXX, registers the numbered status code defined in the ABNF entry "Status-Code" except "extension-code" (that defines the syntax allowed for future extensions) in Section 20.2.2.

22.4. RTSP Headers

22.4.1. Description

By specifying new headers a method(s) can be enhanced in many different ways. An unknown header will be ignored by the receiving agent. If the new header is vital for a certain functionality, a feature-tag for the functionality can be created and demanded to be used by the counter-part with the inclusion of a Require header carrying the feature-tag.

22.4.2. Registering New Headers with IANA

Registrations in the registry can be done following the Expert Review policy. A specification SHOULD be provided, preferable an IETF RFC or other Standards Developing Organization specification. The minimal information in a registration request is the header name and the contact information.

The specification SHOULD contain the following information:

- The name of the header.
- An ABNF specification of the header syntax.
- A list or table specifying when the header may be used, encompassing all methods, their request or response, the direction (C->S or S->C).
- How the header is to be handled by proxies.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1499
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- A description of the purpose of the header.

22.4.3. Registered entries

All headers specified in Section 16 in RFCXXXX are to be registered. The Registry is to include header name, description, and reference.

Furthermore the following RTSP headers defined in other specifications are registered:

- x-wap-profile defined in [3gpp-26234].
- x-wap-profile-diff defined in [3gpp-26234].
- x-wap-profile-warning defined in [3gpp-26234].
- x-predecbufsize defined in [3gpp-26234].
- x-initpredecbufperiod defined in [3gpp-26234].
- x-initpostdecbufperiod defined in [3gpp-26234].
- 3gpp-videopostdecbufsize defined in [3gpp-26234].
- 3GPP-Link-Char defined in [3gpp-26234].
- 3GPP-Adaptation defined in [3gpp-26234].
- 3GPP-QoE-Metrics defined in [3gpp-26234].
- 3GPP-QoE-Feedback defined in [3gpp-26234].

The use of "x-" is NOT RECOMMENDED but the above headers in the register list was defined prior to the clarification.

22.5. Accept-Credentials

The security framework's TLS connection mechanism has two registrable entities.

22.5.1. Accept-Credentials policies

In Section 19.3.1 three policies for how to handle certificates are specified. Further policies may be defined and MUST be registered with IANA using the following rules:

- Registering requires an IETF Standards Action

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1500
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- A registration is required to name a contact person.
- Name of the policy.
- A describing text that explains how the policy works for handling the certificates.

This specification registers the following values:

Any

Proxy

User

22.5.2. Accept-Credentials hash algorithms

The Accept-Credentials header (See Section 16.2) allows for the usage of other algorithms for hashing the DER records of accepted entities. The registration of any future algorithm is expected to be extremely rare and could also cause interoperability problems. Therefore the bar for registering new algorithms is intentionally placed high.

Any registration of a new hash algorithm MUST fulfill the following requirement:

- Follow the IETF Standards Action policy.
- A definition of the algorithm and its identifier meeting the "token" ABNF requirement.

The registered value is:

Hash Alg. Id	Reference
sha-256	[RFCXXXX]

22.6. Cache-Control Cache Directive Extensions

There exist a number of cache directives which can be sent in the Cache-Control header. A registry for these cache directives MUST be defined with the following rules:

- Registering requires an IETF Standards Action or IESG Approval.
- A registration is required to contain a contact person.
- Name of the directive and a definition of the value, if any.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1501
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Specification if it is an request or response directive.
- A describing text that explains how the cache directive is used for RTSP controlled media streams.

This specification registers the following values:

no-cache:

public:

private:

no-transform:

only-if-cached:

max-stale:

min-fresh:

must-revalidate:

proxy-revalidate:

max-age:

The registry should be represented as: Name of the directive, contact person and reference.

22.7. Media Properties

22.7.1. Description

The media streams being controlled by RTSP can have many different properties. The media properties required to cover the use cases that was in mind when writing the specification are defined. However, it can be expected that further innovation will result in new use cases or media streams with properties not covered by the ones specified here. Thus new media properties can be specified. As new media properties may need a substantial amount of new definitions to correctly specify behavior for this property the bar is intended to be high.

22.7.2. Registration Rules

Registering new media property MUST fulfill the following requirements

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1502
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

- Follow the Specification Required policy and get the approval of the designated Expert.
- Have an ABNF definition of the media property value name that meets "media-prop-ext" definition
- A Contact Person for the Registration
- Description of all changes to the behavior of the RTSP protocol as result of these changes.

22.7.3. Registered Values

This specification registers the 9 values listed in Section 16.28. The registry should be represented as: Name of the media property, contact person and reference.

22.8. Notify-Reason header

22.8.1. Description

Notify-Reason values are used for indicating the reason the notification was sent. Each reason has its associated rules on what headers and information that may or must be included in the notification. New notification behaviors need to be specified to enable interoperable usage, thus a specification of each new value is required.

22.8.2. Registration Rules

Registrations for new Notify-Reason value MUST fulfill the following requirements

- Follow the Specification Required policy and get the approval of the designated Expert.
- Have a ABNF definition of the Notify reason value name that meets "Notify-Reason-extension" definition
- A Contact Person for the Registration
- Description of which headers shall be included in the request and response, when it should be sent, and any effect it has on the server client state.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1503
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.8.3. Registered Values

This specification registers 3 values defined in the Notify-Reas-val ABNFSection 20.2.3:

- end-of-stream
- media-properties-update
- scale-change

The registry entries should be represented in the registry as: Name, short description, contact and reference.

22.9. Range header formats

22.9.1. Description

The Range header (Section 16.38) allows for different range formats. New ones may be registered, but moderation should be applied as it makes interoperability more difficult.

22.9.2. Registration Rules

A registration MUST fulfill the following requirements:

- Follow the Specification Required policy.
- An ABNF definition of the range format that fulfills the "range-ext" definition.
- A Contact person for the registration.
- Rules for how one handles the range when using a negative Scale.

22.9.3. Registered Values

The registry should be represented as: Name of the range format, contact person and reference. This specification registers the following values.

npt: Normal Play Time

clock: UTC Clock format

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1504
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

smpte: SMPTE Timestamps

22.10. Terminate-Reason Header

The Terminate-Reason header (Section 16.50) has two registries for extensions.

22.10.1. Redirect Reasons

Registrations are done under the policy of Expert Review. The registered value needs to follow syntax, i.e. be a token. The specification needs to provide definition of what the procedures that is to be followed when a client receives this redirect reason. This specification registers two values:

- Session-Timeout
- Server-Admin

The registry should be represented as: Name of the Redirect Reason, contact person and reference.

22.10.2. Terminate-Reason Header Parameters

Registrations are done under the policy of Specification Required. The registrations must define a syntax for the parameter that also follows the allowed by the RTSP 2.0 specification. A contact person is also required. This specification registers:

- time
- user-msg

The registry should be represented as: Name of the Terminate Reason, contact person and reference.

22.11. RTP-Info header parameters

22.11.1. Description

The RTP-Info header (Section 16.43) carries one or more parameter value pairs with information about a particular point in the RTP stream. RTP extensions or new usages may need new types of information. As RTP information that could be needed is likely to be generic enough and to maximize the interoperability registration requires specification required.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1505
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.11.2. Registration Rules

Registrations for new Notify-Reason value MUST fulfill the following requirements

- Follow the Specification Required policy and get the approval of the designated Expert.
- Have a ABNF definition that meets the "generic-param" definition
- A Contact Person for the Registration

22.11.3. Registered Values

This specification registers 2 parameter value pairs:

- seq
- rtptime

The registry should be represented as: Name of the parameter, contact person and reference.

22.12. Seek-Style Policies

22.12.1. Description

New seek policies may be registered, however, a large number of these will complicate implementation substantially. The impact of unknown policies is that the server will not honor the unknown and use the server default policy instead.

22.12.2. Registration Rules

Registrations of new Seek-Style polices MUST fulfill the following requirements

- Follow the Specification Required policy.
- Have a ABNF definition of the Seek-Style policy name that meets "Seek-S-value-ext" definition
- A Contact Person for the Registration
- Description of which headers shall be included in the request and response, when it should be sent, and any affect it has on the server client state.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1506
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.12.3. Registered Values

This specification registers 4 values:

- RAP
- CoRAP
- First-Prior
- Next

The registry should be represented as: Name of the Seek-Style Policy, short description, contact person and reference.

22.13. Transport Header Registries

The transport header contains a number of parameters which have possibilities for future extensions. Therefore registries for these needs to be defined.

22.13.1. Transport Protocol Specification

A registry for the parameter transport-protocol specification MUST be defined with the following rules:

- Registering uses the policy of Specification Required.
- A contact person or organization with address and email.
- A value definition that are following the ABNF syntax definition of "transport-id" Section 20.2.3.
- A describing text that explains how the registered value are used in RTSP.

The registry should be represented as: The protocol ID string, contact person and reference.

This specification registers the following values:

RTP/AVP: Use of the RTP [RFC3550] protocol for media transport in combination with the "RTP profile for audio and video conferences with minimal control" [RFC3551] over UDP. The usage is explained in RFC XXXX, Appendix C.1.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1507
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RTP/AVP/UDP: the same as RTP/AVP.

RTP/AVPF: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended RTP Profile for RTCP-based Feedback (RTP/AVPF)" [RFC4585] over UDP. The usage is explained in RFC XXXX, Appendix C.1.

RTP/AVPF/UDP: the same as RTP/AVPF.

RTP/SAVP: Use of the RTP [RFC3550] protocol for media transport in combination with the "The Secure Real-time Transport Protocol (SRTP)" [RFC3711] over UDP. The usage is explained in RFC XXXX, Appendix C.1.

RTP/SAVP/UDP: the same as RTP/SAVP.

RTP/SAVPF: Use of the RTP[RFC3550] protocol for media transport in combination with the Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF) [RFC5124] over UDP. The usage is explained in RFC XXXX, Appendix C.1.

RTP/SAVPF/UDP: the same as RTP/SAVPF.

RTP/AVP/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "RTP profile for audio and video conferences with minimal control" [RFC3551] over TCP. The usage is explained in RFC XXXX, Appendix C.2.2.

RTP/AVPF/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended RTP Profile for RTCP-based Feedback (RTP/AVPF)" [RFC4585] over TCP. The usage is explained in RFC XXXX, Appendix C.2.2.

RTP/SAVP/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "The Secure Real-time Transport Protocol (SRTP)" [RFC3711] over TCP. The usage is explained in RFC XXXX, Appendix C.2.2.

RTP/SAVPF/TCP: Use of the RTP [RFC3550] protocol for media transport in combination with the "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] over TCP. The usage is explained in RFC XXXX, Appendix C.2.2.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1508
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

22.13.2. Transport modes

A registry for the transport parameter mode MUST be defined with the following rules:

- Registering requires an IETF Standards Action.
- A contact person or organization with address and email.
- A value definition that are following the ABNF "token" definition Section 20.2.3.
- A describing text that explains how the registered value are used in RTSP.

This specification registers 1 value:

PLAY: See RFC XXXX.

22.13.3. Transport Parameters

A registry for parameters that may be included in the Transport header MUST be defined with the following rules:

- Registering uses the Specification Required policy.
- A value definition that are following the ABNF "token" definition Section 20.2.3.
- A describing text that explains how the registered value are used in RTSP.

This specification registers all the transport parameters defined in Section 16.52.

22.14. URI Schemes

This specification defines two URI schemes ("rtsp" and "rtsps") and reserves a third one ("rtspu"). Registrations are following RFC 4395[RFC4395].

22.14.1. The rtsp URI Scheme

URI scheme name: rtsp

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1509
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Status: Permanent

URI scheme syntax: See Section 20.2.1 of RFC XXXX.

URI scheme semantics: The rtsp scheme is used to indicate resources accessible through the usage of the Real-time Streaming Protocol (RTSP). RTSP allows different operations on the resource identified by the URI, but the primary purpose is the streaming delivery of the resource to a client. However, the operations that are currently defined are: Describing the resource for the purpose of configuring the receiving agent (DESCRIBE), configuring the delivery method and its addressing (SETUP), controlling the delivery (PLAY and PAUSE), reading or setting of resource related parameters (SET_PARAMETER and GET_PARAMETER, and termination of the session context created (TEARDOWN).

Encoding considerations: IRIs in this scheme are defined and needs to be encoded as RTSP URIs when used within the RTSP protocol. That encoding is done according to RFC 3987.

Applications/protocols that use this URI scheme name: RTSP 1.0 (RFC 2326), RTSP 2.0 (RFC XXXX)

Interoperability considerations: The change in URI syntax performed between RTSP 1.0 and 2.0 can create interoperability issues.

Security considerations: All the security threats identified in Section 7 of RFC 3986 applies also to this scheme. They need to be reviewed and considered in any implementation utilizing this scheme.

Contact: Magnus Westerlund, magnus.westerlund@ericsson.com

Author/Change controller: IETF

References: RFC 2326, RFC 3986, RFC 3987, RFC XXXX

22.14.2. The rtsps URI Scheme

URI scheme name: rtsps

Status: Permanent

URI scheme syntax: See Section 20.2.1 of RFC XXXX.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1510
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

URI scheme semantics: The rtsps scheme is used to indicate resources accessible through the usage of the Real-time Streaming Protocol (RTSP) over TLS. RTSP allows different operations on the resource identified by the URI, but the primary purpose is the streaming delivery of the resource to a client. However, the operations that are currently defined are: Describing the resource for the purpose of configuring the receiving agent (DESCRIBE), configuring the delivery method and its addressing (SETUP), controlling the delivery (PLAY and PAUSE), reading or setting of resource related parameters (SET_PARAMETER and GET_PARAMETER, and termination of the session context created (TEARDOWN).

Encoding considerations: IRIs in this scheme are defined and needs to be encoded as RTSP URIs when used within the RTSP protocol. That encoding is done according to RFC 3987.

Applications/protocols that use this URI scheme name: RTSP 1.0 (RFC 2326), RTSP 2.0 (RFC XXXX)

Interoperability considerations: The change in URI syntax performed between RTSP 1.0 and 2.0 can create interoperability issues.

Security considerations: All the security threats identified in Section 7 of RFC 3986 applies also to this scheme. They need to be reviewed and considered in any implementation utilizing this scheme.

Contact: Magnus Westerlund, magnus.westerlund@ericsson.com

Author/Change controller: IETF

References: RFC 2326, RFC 3986, RFC 3987, RFC XXXX

22.14.3. The rtspu URI Scheme

URI scheme name: rtspu

Status: Permanent

URI scheme syntax: See Section 3.2 of RFC 2326.

URI scheme semantics: The rtspu scheme is used to indicate resources accessible through the usage of the Real-time Streaming Protocol (RTSP) over unreliable datagram transport. RTSP allows different operations on the resource identified by the URI, but the primary purpose is the streaming delivery of the resource to a client. However, the operations that are

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1511
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

currently defined are: Describing the resource for the purpose of configuring the receiving agent (DESCRIBE), configuring the delivery method and its addressing (SETUP), controlling the delivery (PLAY and PAUSE), reading or setting of resource related parameters (SET_PARAMETER and GET_PARAMETER, and termination of the session context created (TEARDOWN).

Encoding considerations: IRIs in this scheme are not defined.

Applications/protocols that use this URI scheme name: RTSP 1.0 (RFC 2326)

Interoperability considerations: The definition of the transport mechanism of RTSP over UDP has interoperability issues. That makes the usage of this scheme problematic.

Security considerations: All the security threats identified in Section 7 of RFC 3986 applies also to this scheme. They needs to be reviewed and considered in any implementation utilizing this scheme.

Contact: Magnus Westerlund, magnus.westerlund@ericsson.com

Author/Change controller: IETF

References: RFC 2326

22.15. SDP attributes

This specification defines three SDP [RFC4566] attributes that it is requested that IANA register.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1512
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

SDP Attribute ("att-field"):

Attribute name:	range
Long form:	Media Range Attribute
Type of name:	att-field
Type of attribute:	Media and session level
Subject to charset:	No
Purpose:	RFC XXXX
Reference:	RFC XXXX, RFC 2326
Values:	See ABNF definition.

Attribute name:	control
Long form:	RTSP control URI
Type of name:	att-field
Type of attribute:	Media and session level
Subject to charset:	No
Purpose:	RFC XXXX
Reference:	RFC XXXX, RFC 2326
Values:	Absolute or Relative URIs.

Attribute name:	mtag
Long form:	Message Tag
Type of name:	att-field
Type of attribute:	Media and session level
Subject to charset:	No
Purpose:	RFC XXXX
Reference:	RFC XXXX
Values:	See ABNF definition

22.16. Media Type Registration for text/parameters

Type name: text

Subtype name: parameters

Required parameters:

Optional parameters:

Encoding considerations:

Security considerations: This format may carry any type of parameters. Some can clear have security requirements, like privacy, confidentiality or integrity requirements. The format has no built in security protection. For the usage it was defined the transport can be protected between server and client using TLS. However, care must be take to consider if also the proxies

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1513
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

are trusted with the parameters in case hop-by-hop security is used. If stored as file in file system the necessary precautions needs to be taken in relation to the parameters requirements including object security such as S/MIME [RFC5751].

Interoperability considerations: This media type was mentioned as a fictional example in RFC 2326 but was not formally specified. This have resulted in usage of this media type which may not match its formal definition.

Published specification: RFC XXXX, Appendix F.

Applications that use this media type: Applications that use RTSP and have additional parameters they like to read and set using the RTSP GET_PARAMETER and SET_PARAMETER methods.

Additional information:

Magic number(s):

File extension(s):

Macintosh file type code(s):

Person & email address to contact for further information: Magnus Westerlund (magnus.westerlund@ericsson.com)

Intended usage: Common

Restrictions on usage: None

Author: Magnus Westerlund (magnus.westerlund@ericsson.com)

Change controller: IETF

Addition Notes:

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1514
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

23. References

23.1. Normative References

[3gpp-26234]
Third Generation Partnership Project (3GPP), "Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs; Technical Specification 26.234", December 2002.

[FIPS-pub-180-2]
National Institute of Standards and Technology (NIST), "Federal Information Processing Standards Publications (FIPS PUBS) 180-2: Secure Hash Standard", August 2002.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)",

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1515
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

RFC 3711, March 2004.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC4567] Arkko, J., Lindholm, F., Naslund, M., Norrman, K., and E. Carrara, "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)", RFC 4567, July 2006.

[RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, July 2006.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4738] Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1516
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

Multimedia Internet KEYing (MIKEY)", RFC 4738, November 2006.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

23.2. Informative References

[I-D.ietf-mmusic-rtsp-nat]
Goldberg, J., Westerlund, M., and T. Zeng, "A Network Address Translator (NAT) Traversal mechanism for media controlled by Real-Time Streaming Protocol (RTSP)", draft-ietf-mmusic-rtsp-nat-09 (work in progress), January 2010.

[ISO.13818-6.1995]
International Organization for Standardization, "Information technology - Generic coding of moving pictures and associated audio information - part 6: Extension for digital storage media and control", ISO Draft Standard 13818-6, November 1995.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1517
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[ISO.8601.2000]
International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO/IEC Standard 8601, December 2000.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC1644] Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.

[RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, July 2009.

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1518
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)

[Stevens98]
Stevens, W., "Unix Networking Programming - Volume 1, second edition", 1998.

Netflix, Inc. and Hulu, LLC - Ex. 1002, Page 1519
IPR2020-00558 (Netflix, Inc. and Hulu, LLC v. DivX, LLC)