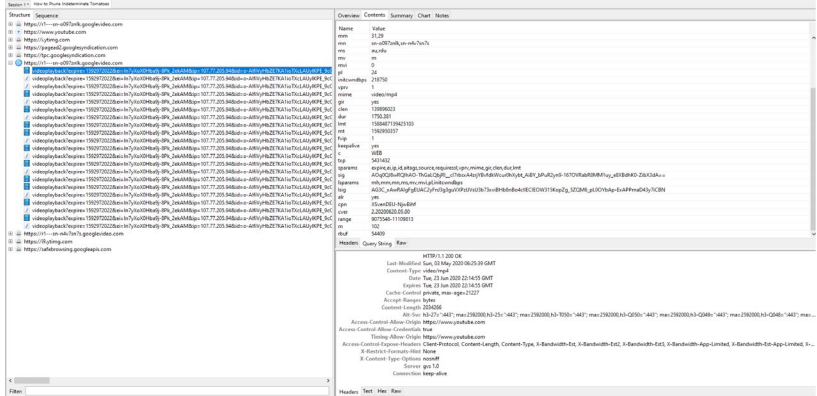
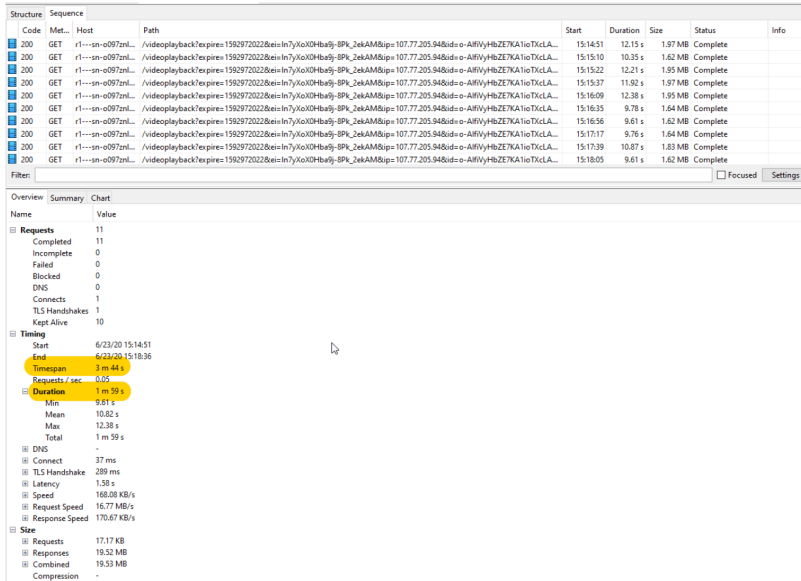


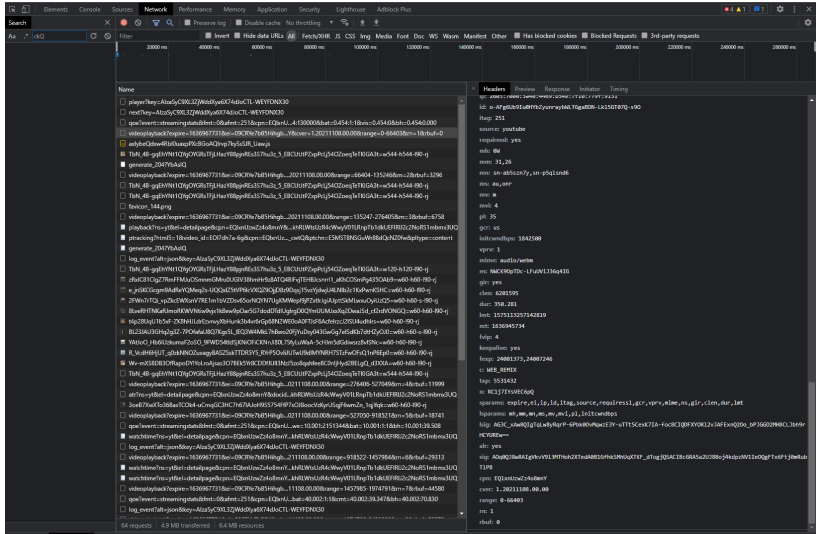
Claim Language	Contention
Overview	WAG has asserted three patents against YouTube, which concern streaming media over the Internet, conducted between a user (client) system, which requests and plays the media, and a server to which the client system connects over the Internet, which provides the requested media. WAG's '824 and '636 patents concern methods, apparatus, and computer program products on the server side of this interaction, for prerecorded and live media, respectively (which differ, in that the latter concerns media created contemporaneously with its streaming). WAG's '594 patent (the subject of the within contentions) concerns methods, apparatus, and computer program products on the client side of the above-described interaction, which commonly apply whether the server is providing prerecorded or live media. Because the subject matter of these patents overlap, WAG's infringement contentions with regard to the '824 and '636 patents are each incorporated herein by reference. YouTube LLC and/or affiliated entities such as Google LLC ("YouTube") directs and controls the activity of streaming digitally encoded media it provides over the Internet through the site youtube.com (also accessible as www.youtube.com) (the "YouTube Web Site") from servers owned and/or the operations of which are controlled by YouTube, including servers in the U.S. and in the Western District of Texas ("YouTube's Servers"). In one mode of operation, YouTube distributes pre-recorded video programs from YouTube's servers, over the Internet, to large numbers of users in the U.S. and elsewhere. This is done via a "pull" protocol (as further addressed herein), defined at the application layer and generally conforming with the MPEG-DASH specification, ISO/IEC 23009 (citations are to the fourth edition, Dec. 2009 (cited herein as "ISO/IEC 23009"), though WAG does not believe that any versions applicable during the period of the alleged infringement had any material differences). Google is a Charter Member of the DASH Industry Forum. A protocol similar to MPEG-DASH is HTTP Live Streaming ("HLS"). HLS is a streaming media delivery specification, currently published as IETF RFC 8216. This claim chart is based on evidence observed in the course of YouTube's streaming. This claim chart alleges infringement based on such observations, without depending on whether YouTube fully complied with the MPEG-DASH or HLS standards in performing such streaming. However, WAG also asserts that the fact of compliance with either of those standards in itself separately supports all claim limitations and the conclusion of infringement. WAG has separately charted infringement of the '824, '594, and '636 patents that results from complying with the MPEG-DASH and HLS standards, and incorporates those charts herein by reference. The evidence herein is consistent with full compliance by YouTube with the MPEG-DASH standard from 2017-2021, and WAG believes YouTube fully complied with that standard during such period. Google's implementation of MPEG-DASH has been the subject of some academic work, including the paper Mondal, et al., "Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption," Proceedings of the

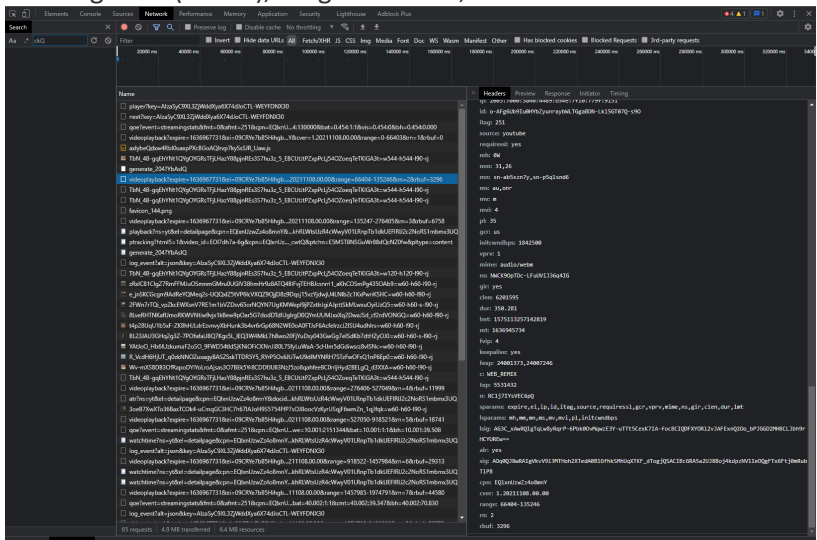
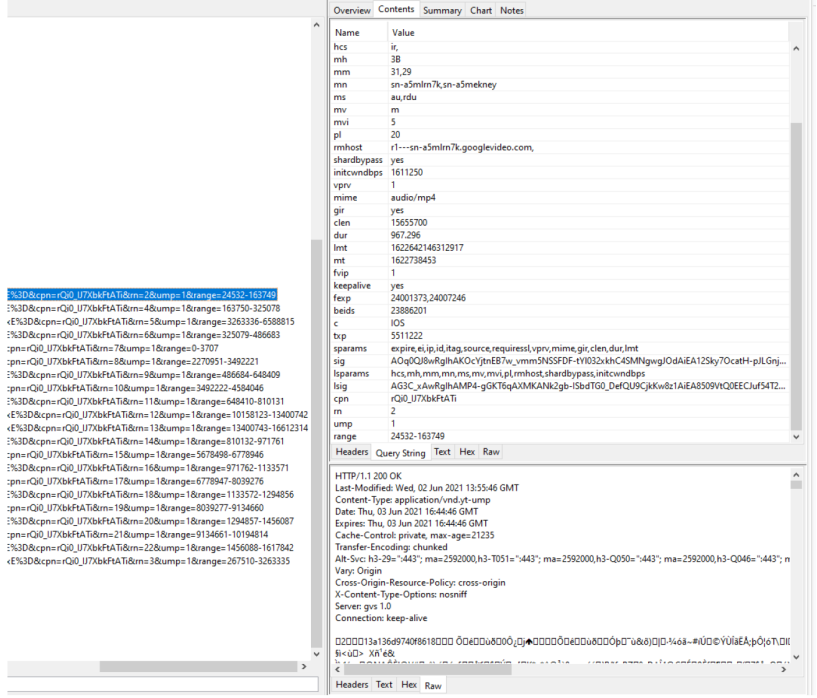
Claim Language	Contention
	Workshop on Network and Operating Systems Support for Digital Audio and Video: 19-24 Taipei, Taiwan: Association for Computing Machinery, doi: 10.1145/3083165.3083177 (2017), source: https://dl.acm.org/citation.cfm?doid=3083165.3083177 isbn: 978-1-4503-5003-7. WAG intends to further investigate of YouTube’s compliance with the MPEG-DASH standard in discovery. WAG also has specific evidence of use of HLS by YouTube for certain live streams (see ‘636 claim chart), but currently has no direct evidence of HLS having been used by YouTube from 2017-2021 for pre-recorded media. WAG has been impeded from investigating such usage in part by the fact that YouTube streams are often encrypted. However, there is evidence that YouTube has streamed with HLS beyond the specific types of sessions currently documented in these claim charts. For example, YouTube’s web pages (including those for pre-recorded media) also contain reference to a manifest for HLS, in addition to providing a link to an MPEG-DASH manifest. The client-side JavaScript files that YouTube distributes in support of its streaming media (e.g., base.js) also provide functions for HLS. YouTube web pages also provide links to HLS manifests (as well as MPEG-DASH manifests). HLS use by YouTube is further reflected in YouTube’s technical support pages, e.g., https://support.google.com/youtube/answer/2853702?hl=en&ref_topic=2853712 and https://support.google.com/youtube/answer/10349430. WAG contends that YouTube has supported streaming media delivery of pre-recorded programs during the relevant period via HLS as at least a fallback where the user device identifies itself as a device that normally will not support MPEG-DASH or prefers HLS. WAG intends to pursue discovery to investigate the foregoing and update these contentions as necessary. Under the MPEG-DASH standard (ISO/IEC 23009 at ix), as well as HLS (§ 1 at 4), control of streaming media delivery lies primarily with the client, and YouTube’s accused streaming reflects the same characteristic, sometimes referred to as “Pull.” The server is responsible for dividing, storing, and identifying streaming content data elements to be responsive to requests it will receive from clients, but the timing and mechanics of delivering the requested elements is determined by processes that run on the client. Within the limits of the ability of the server and the communication channel to serve requested elements, the pacing of streaming delivery is controlled by the client. To make such a “pull” delivery scheme work, YouTube must ensure that software that is compatible with the manner in which the content is stored on YouTube’s servers is operable on the client side. YouTube does this in two different ways. First, for user-agents (clients) that can execute JavaScript or a comparable language, YouTube provides from its servers JavaScript files that implement the desired content-requesting functionality. This can be done as part of the streaming session. For other user-agents, such as embedded user-agents (e.g., various Roku devices, tvOS, FireOS, iOS, Android, etc., as well as native iPhone and iPad, YouTube provides a comparable native application for the device, for example, through the device provider’s app store, or built into the device. In the case of the iPhone, the

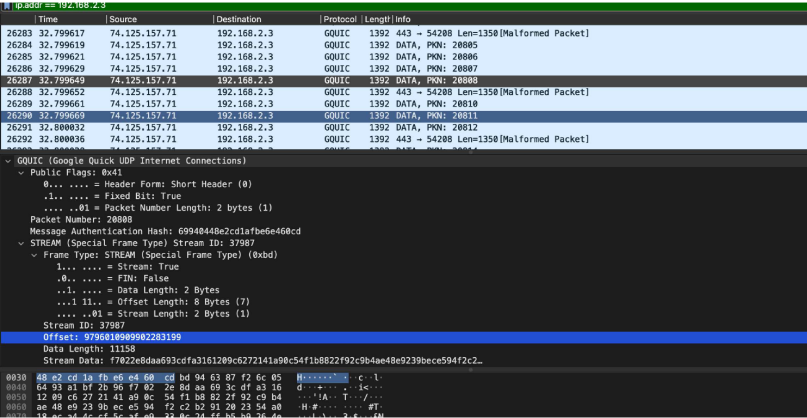
Claim Language	Contention
	Safari browser provided with the device lacked Media Source Extensions (“MSE”). Streaming protocols other than Safari-supported HLS (or download) required a separate app (e.g., the YouTube iPhone App). However, in some cases the iPhone could still stream some Youtube content using the Safari browser, without loading a separate YouTube app. These contentions relate to every streaming video client-side program provided through the YouTube Site or by YouTube as integrated into devices it makes or sells, or provided by YouTube to app stores for the respective devices, for the period from the 2017 issuance date of the patent addressed in these contentions, through the common date of expiration of the patent (March 28, 2021). The contentions apply to the software and devices streaming either pre-recorded or live video. Also included in these contentions is streaming audio distribution from the YouTube TV subscription service (available for distribution through tv.youtube.com), and music.youtube.com (formerly Google Play Music). The foregoing services are addressed individually, but are also referred to, for purposes of collective reference, as the “YouTube Streaming Services.” The web debugging proxy described in connection with the ‘824 patent is also relevant to the ‘594 patent, in that, for purposes of investigation, the client device and its software relevant to the ‘594 patent sits behind the proxy. Alternately (as also described in the 824 contentions), built-in browser “developer tools” or “console,” with browsers such as Chrome, Firefox, and desktop versions of Safari could also be used to observe and capture client-server interactions for YouTube videos. As seen with respect to the ‘824 patent, a streaming session involves a set of requests/responses illustrated by the following actual example of a client-server request/response sequence for a program on YouTube as captured in June 2020 by a web debugging proxy:  The entirety of each requested video program (from beginning to end) is delivered as a series of time-sequenced media data elements, where each such element has been given one or more serial identifiers (e.g., range=9075548-11109813, rn=102), and requested by serial identifier, as reflected by the sample sequence above, reflecting 11 such successive requests, interleaved with similarly sequenced requests for the audio portion of the program.

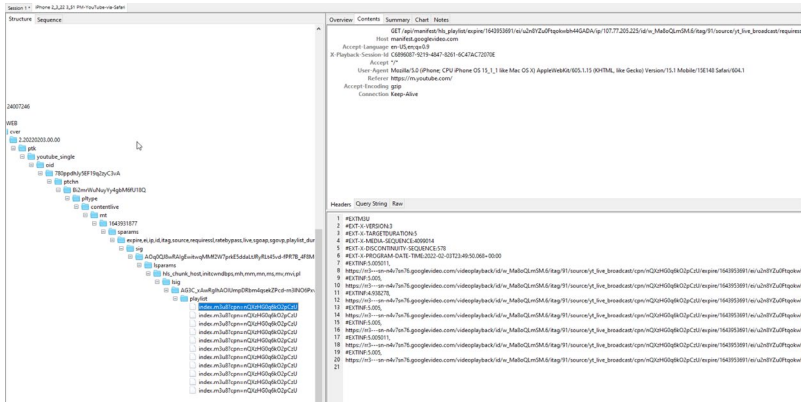
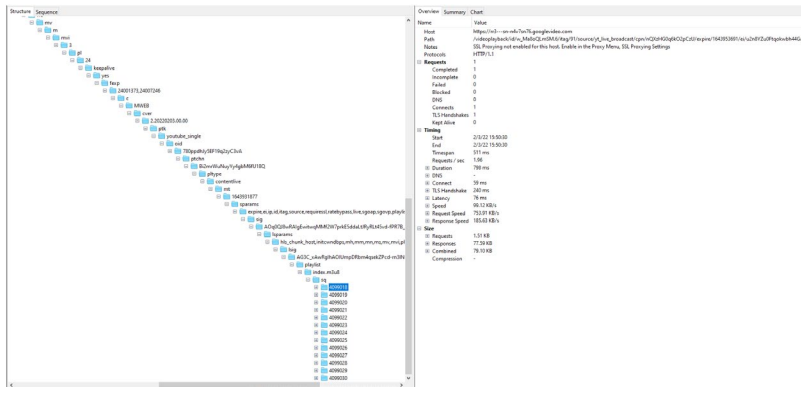
Claim Language	Contention
	The client-side software in these examples is responsible for requesting each such element, by its serial identifier, when and as needed by the client, in order to sustain continuous and uninterrupted playback. Because the streaming media flow as to each element in the stream is thus dependent on specific client-side requests for the respective elements comprising the stream, the overall process is sometimes referred to as a “pull.” Furthermore, in order for the client to successfully keep up with the stream while making such pull requests, it should not take longer to request and receive the individual segments than the time it takes to play the segments back at a normal rendition. Repeated observations confirm that transmissions from YouTube Servers consistently meet these requirements. For example, in the below sequence, requesting and receiving video data having 3m 44 sec of playback time only uses 1m 59sec. for the transfer, meaning that the data is transferred in about half the time it takes for the client system to play it back:  The MPEG-DASH standard, ISO/IEC 23009, spells out media distribution steps that align identically with each of the claim limitations herein. The version of the MPEG-DASH specification referenced herein is the Fourth edition, Dec. 2019. However, WAG believes that YouTube also complied with earlier and later versions of this standard, with no differences between such versions being material to the claims herein. YouTube’s HLS streams also reflect compliance with the applicable standard, IETF RFC 8216, <i>e.g.</i>, the version of this RFC published in Aug. 2017. The foregoing figure is representative of every platform (whether MPEG-DASH or HLS) streaming from YouTube Servers for which WAG has been able to observe the contents of HTTP(S) traffic, including Chrome, Firefox, Safari, and other desktop browsers, iOS on iPhone and iPad, and Android, as well as TVOS and FireOS. It is also representative of Google Play Music/YouTube Music, except insofar as these audio services are limited to audio media data elements (still nevertheless internally named “videoplayback”), as well as YouTube TV.

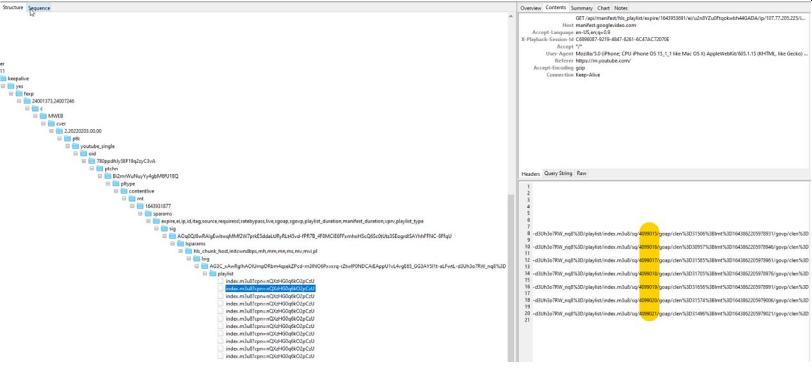
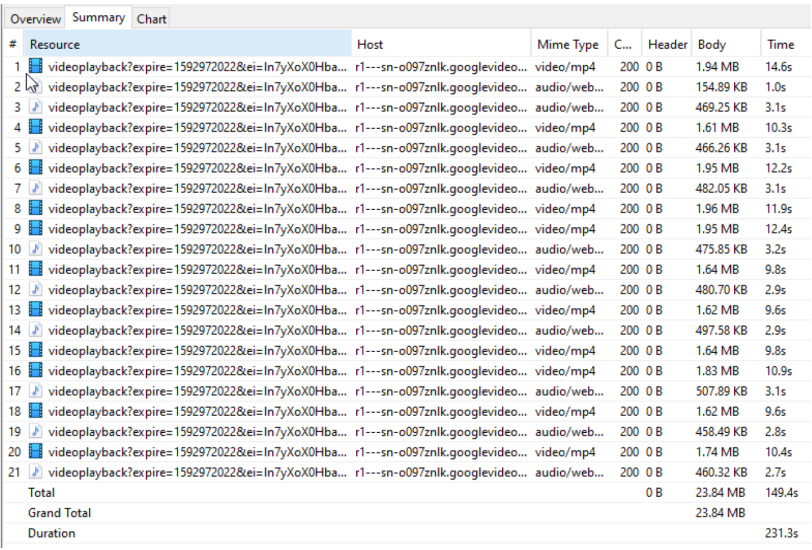
Claim Language	Contention
	On some embedded platforms (such as Roku and TVOS), due to encryption, WAG has been unable to inspect the contents of HTTPS (encrypted) requests and responses. However, the timing of the encrypted requests and responses follow the same pattern as observable on other platforms (e.g., desktop Chrome, as well as the FireOS embedded platform), and WAG believes the MPEG-DASH streaming implementation is the same in all material respects to the other observed implementations. WAG will include in its discovery requests inquiries directed to the additional platforms whose communications have been obscured by encryption. In the case of JavaScript implementations, the code is downloaded into the browser, as a file named, for example, base.js. WAG believes, based on the similarity in the timing and data transfers when viewing the same programs, that YouTube-provided embedded apps behave in the same manner as the JavaScript in all material respects. WAG will request discovery of the source code for YouTube apps to allow it to further develop these contentions with regard to the range of YouTube client software formats.
[1.P.1] A method for operating a media player to receive and play an audio or video program, from a remote media source via a data connection over the Internet,	This method is performed when operating a client to access YouTube streaming content.
[1.T] the method comprising:	
[1.1.1] sending requests from the media player to the media source via the data connection	The above capture (and others reflected below) shows a plurality of HTTP GET requests made from a media player (desktop computer running the Chrome browser) to an identified media source on YouTube, via the player's data connection to the server. This plurality of requests from the media player to the media source is also reflected in a manifest. YouTube's MPEG-DASH videos are requested by the user device, by successive segments of video and audio, pursuant to a manifest supplied to the user device by the YouTube server. Such manifests contain entries such as the following, indicating storage of the successive audio and video segments in a file system on the YouTube servers: <pre><SegmentList> <Initialization sourceURL="range/0-258"/> <SegmentURL media="range/460-187101"/> <SegmentURL media="range/187102-376159"/> <SegmentURL media="range/376160-561636"/> <SegmentURL media="range/561637-745450"/> <SegmentURL media="range/745451-926622"/> <SegmentURL media="range/926623-1114800"/> <SegmentURL media="range/1114801-1299024"/> <SegmentURL media="range/1299025-1481544"/></pre>

Claim Language	Contention
	<SegmentURL media="range/1481545-1664235"/> <SegmentURL media="range/1664236-1844136"/> <SegmentURL media="range/1844137-2020117"/> <SegmentURL media="range/2020118-2151945"/> </SegmentList> The above is from the manifest for the video at https://www.youtube.com/watch?v=i2lhwb_OckQ (a different video, which also reflects a series of requests/responses by range and rn as shown in the first example above). The manifest is designed, per the MPEG-DASH specification, to cause the user device to sequentially request the listed media segments, resulting in the continuous playback of the subject video. The right side of each of the above expressions represents a path to a storage location of the corresponding segment, on the YouTube server, wherein “range” represents a directory (folder), and “[number]-[number]” represents a file (or an equivalent to a folder-file arrangement, as implemented in a database or other data structure). The JavaScript or equivalent user-side program causes the media player to iterate through the provided list of segments to request, over its data connection, the segments comprising the desired stream, from the media source.
[1.1.2] for one or more serially identified media data elements representing the audio or video program,	<i>E.g.</i>, the shown request for bytes 9075548-11109813 and rn 102 represent a portion of a stream for the requested program. This pattern, wherein the client drives the streaming process by making successive requests by serial identifiers to a server for a sequence of segments that comprise an audio or video program is repeated over the entire range of on-demand and live streaming from YouTube. The following are corresponding request-response timelines for other platforms streaming from YouTube Streaming Services: Streaming from Google Music to a desktop computer (Rod Stewart song Maggie May, requests/responses captured in Chrome Developer Console on a desktop computer: 

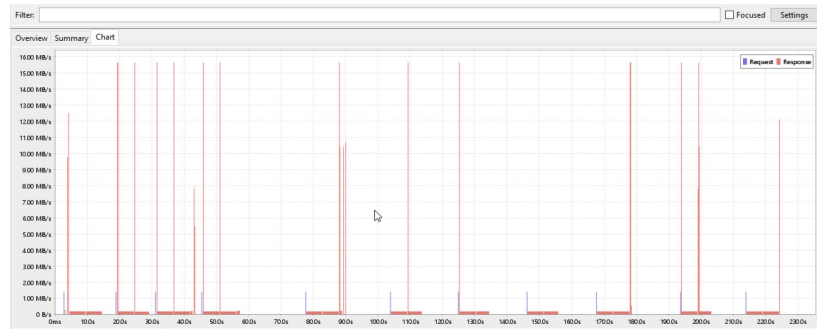
Claim Language	Contention
	First segment (above), range: 0-66403, rn: 1  Second segment, range: 66404-1352468, rn: 2 (both next in sequence); the following video playback elements follow the same pattern. Streaming to an iOS device:  YouTube on iOS (as captured by a proxy) shows the same patterns, with successively incrementing range and rn identifiers for the respective media segments. Other Embedded Devices

Claim Language	Contention
	WAG has also been able to intercept successive requests and responses of characteristic length for streaming YouTube media to tvOS and FireOS devices, but encryption has prevented WAG from seeing the identifiers exchanged. For example, a packet capture for a YouTube streaming session from tvOS showed the following:  WAG believes these embedded platforms (e.g., tvOS, FireOS, etc., as well as Roku during the relevant period) operated with YouTube in substantially the same way as observed for desktop and iOS devices. WAG will include in its discovery requests inquiries directed to the additional platforms whose communications have been obscured by encryption. YouTube’s distribution of live programming (to the same array of consuming devices) follows the same pattern. The YouTube Live transmissions, covered in the ’636 patent claim chart, have “sq” numbers in the requests (in addition to rn), which are also sequential. YouTube’s JavaScript and Apps likewise infringe the ’594 patent for live media. For live programming, the sq numbers are used in the manifest to indicate a directory for the element files corresponding to the byte ranges. See corresponding contentions for the ’636 patent, incorporated herein by reference. The “range” and “rn” values, both individually and as an ordered pair, literally provide serial identification of the media data elements. Nonetheless, this limitation may also be found under the doctrine of equivalents. In particular, the function of serially identifying the media data elements is to uniquely identify the time-sequenced media data elements from each other, which is exactly what the “range” and “rn” values do, both individually and as an ordered pair. The way in which the media data elements are uniquely identified is to provide each with a unique identifier that can be used in requests from the client to the server; the unique “range” and “rn” values associated with each media data element perform this function, both individually and as an ordered pair. The result of the serial identification is that the client and server agree upon identifications that allow the client to uniquely request a media data element in the proper time sequence, which is what occurs when the client uses the “range” and/or “rn” values to specify a desired media data element. In HLS transmissions (for example, YouTube live programming to Safari on the iPhone), the serially identified elements may be seen in the manifest.

Claim Language	Contention
	The following shows a manifest from such a session:  The serial identifiers are encoded into the paths of the element URLs listed in the manifest. However, the path strings are too long to see in full in the above illustration. The following is an example of the full string for one of the paths shown above: https://rr3---sn-n4v7sn76.googlevideo.com/videoplayback/id/w_Ma8oQLmSM.6/itag/91/source/yt_live_broadcast/cpn/nQxzHG0q6kO2pCzU/expire/1643953691/ei/u2n8YZu0Ftqokwbh44GA DA/ip/107.77.205.225/requirements/yes/ratebypass/yes/live/1/sgoap/gir%3Dyes%3Bitag%3D1 39/sgovp/gir%3Dyes%3Bitag%3D160/hls_chunk_host/rr3---sn-n4v7sn76.googlevideo.com/pla ylist_duration/30/manifest_duration/30/vprv/1/playlist_type/DVR/initcwndbps/1560/mh/ wG/mm/44/mn/sn-n4v7sn76/ms/lva/mv/m/mvi/3/pl/24/keepalive/yes/fexp/24001373,24007246 /c/MWEB/cver/2.20220203.00.00/ptk/youtube_single/oid/780ppdhJy5EF19q2zyC3vA/ptchn/Bi 2mrWuNuyY4gbM6fU18Q/pltype/contentlive/mt/1643931877/sparms/expire,ei,ip,id,itag,s ource,requirements,ratebypass,live,sgoap,sgovp,playlist_duration,manifest_duration,vpr v,playlist_type/sig/AOqOQJ8wRAIgEwltwqMMf2W7prkE5ddaLrYRLt45vd-fPR7B_4F8MCIE6FFxmh oHScQ6Sc0tUt3SEogrDI5AYhhFNC-8FfqU/lsparms/hls_chunk_host/initcwndbps,mh,mm,mn,ms ,mv,mvi,pl/isig/AG3C_xAwRgihAOIUmPDRbm4qsekZPcd-rn3INO6Pxxvr-rZtwlPONDAIEAppU1vL4v gE65_GG3AY5l1t-aLFWtL-d3Uhz07RW_nq8%3D/playlist/index.m3u8/sq/4099018/goap/clen%3D31 705%3Blmt%3D1643862205978976/govp/clen%3D26795%3Blmt%3D1643862205978979/dur/5.005/fi le/seg.ts The paths in successive lines of the manifest are distinguished from each other by an /sq/[number] sequence as shown in the example above. The following are alternate views of this data, showing the sequence in the example stream: 

Claim Language	Contention
	 Though these captures were taken after patent expiration, WAG believes they are representative of corresponding transmissions during the term, insofar as YouTube supported Safari on the iPhone, and Safari has always lacked MSE.
[1.1.3] each media data element comprising a digitally encoded portion of the audio or video program	In the June 2020 capture referenced above (for example), the requested element comprises 2,034,266 bytes, representing a portion (one of a large number requested over the course of receiving the program) of the requested program. The captured sequence was as follows (the above capture for rn 102 (and corresponding range) corresponding to the first line below):  This sequence reflects that the media data element corresponding to rn 102 (and corresponding range) comprises a digitally encoded portion of the requested program, consistent with the fact that each of the media data elements requested by the client comprise a digitally encoded portion of the program being played. In an HLS (live to Safari/iPhone) session, the corresponding data was as follows (approximately 50.5 seconds of playback media received in 12.8 seconds):

Claim Language	Contention																																																																																																																																								
	Sequence Sequence https://manifest.googlevideo.com https://m.youtube.com mp4 youtube https://manifest.googlevideo.com Overview Summary Chart <table><thead><tr><th>#</th><th>Resource</th><th>Host</th><th>MIME Type</th><th>Co.</th><th>Header</th><th>Body</th><th>Time</th></tr></thead><tbody><tr><td>1</td><td>seg1</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>642 B</td><td>76.77 KB</td><td>31 ms</td></tr><tr><td>2</td><td>seg2</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>29 ms</td></tr><tr><td>3</td><td>seg3</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>4</td><td>seg4</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>5</td><td>seg5</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>6</td><td>seg6</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>7</td><td>seg7</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>8</td><td>seg8</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>9</td><td>seg9</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>10</td><td>seg10</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>11</td><td>seg11</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>12</td><td>seg12</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>13</td><td>seg13</td><td>rs--on-nk-7n7g.googlevideo.com</td><td>application/octet-stream</td><td>200</td><td>646 B</td><td>73.63 KB</td><td>1.0s</td></tr><tr><td>Total</td><td></td><td></td><td></td><td></td><td></td><td>803.30 KB</td><td>12.0s</td></tr><tr><td>Grand Total</td><td></td><td></td><td></td><td></td><td></td><td>804.03 KB</td><td></td></tr><tr><td>Duration</td><td></td><td></td><td></td><td></td><td></td><td></td><td>30.5s</td></tr></tbody></table>	#	Resource	Host	MIME Type	Co.	Header	Body	Time	1	seg1	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	642 B	76.77 KB	31 ms	2	seg2	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	29 ms	3	seg3	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	4	seg4	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	5	seg5	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	6	seg6	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	7	seg7	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	8	seg8	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	9	seg9	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	10	seg10	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	11	seg11	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	12	seg12	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	13	seg13	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s	Total						803.30 KB	12.0s	Grand Total						804.03 KB		Duration							30.5s
#	Resource	Host	MIME Type	Co.	Header	Body	Time																																																																																																																																		
1	seg1	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	642 B	76.77 KB	31 ms																																																																																																																																		
2	seg2	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	29 ms																																																																																																																																		
3	seg3	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
4	seg4	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
5	seg5	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
6	seg6	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
7	seg7	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
8	seg8	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
9	seg9	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
10	seg10	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
11	seg11	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
12	seg12	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
13	seg13	rs--on-nk-7n7g.googlevideo.com	application/octet-stream	200	646 B	73.63 KB	1.0s																																																																																																																																		
Total						803.30 KB	12.0s																																																																																																																																		
Grand Total						804.03 KB																																																																																																																																			
Duration							30.5s																																																																																																																																		
[1.1.4] and having a playback rate,	The requested media data element is digitally encoded at a bitrate range and framerate such that a normal rendition occurs when played back at that framerate, as also made evident by the client player that plays back the media data elements at the corresponding playback rate.																																																																																																																																								
[1.2.1] receiving each of the requested media data elements via the data connection,	The rows on the left side of the above capture (and in the summary in 1.1.3) reflects 21 successive requests and responses via the data connection (for audio and video segments in the program, which are interleaved), each, like the request/response on the right (for the highlighted row), showing an HTTP “200 OK” response code (“success”). The success code reflects receipt of the requested element by the client.																																																																																																																																								
[1.2.2] wherein the data connection has a data rate more rapid than the playback rate of the media data elements,	As seen in the overview above, the data for playback is transferred in a fraction of the time required to play it back at normal rendition. It is reasonable to infer, therefore, that the data connection had a data rate in excess of the playback rate.																																																																																																																																								
[1.2.3] and each received media data element is received at a rate as fast as the data connection between the media source and the media player allows;	WAG is not aware of any routine in the client-side code that YouTube supplies for the user devices, that in any way throttles the data connection. Furthermore, WAG’s observation is that the timings reflected in these captures for the reception of the individual elements reflect that the data for each element is being received at the full speed of the data connection. Although this limitation is literally met, it can also be found under the doctrine of equivalents. The function of this receiving step is to ensure that the media data elements are received by the user system fast enough to prevent interruptions to the playback on the user device; client devices receive media data elements from YouTube’s servers, or the CDN servers under YouTube’s control, fast enough to prevent interruptions to playback, as evidenced by the uninterrupted playback on the client end. The way in which this receiving is performed is by receiving the media data elements from the data connection that has a data rate that is more rapid than the playback rate of the media data elements sent via that connection, which, as indicated above in 1.2.2, is what the client does. The result is uninterrupted playing of the time sequenced media data elements on the client device, which, as indicated, is observed on the client device.																																																																																																																																								
[1.3] storing the received media data elements in a memory of the media player;	The client-side code provided by YouTube performs the function of storing each received element in a memory of the media player. This is also confirmed by observing that if the data connection is disrupted (for example, the network cable unplugged on a cabled connection), playback continues for a period of time (but																																																																																																																																								

Claim Language	Contention
	generally not through to the end of the program), reflecting the media data elements were stored in a memory of the media player.
[1.4] playing the received media data elements in series from the memory of the media player; and	The client-side code provided by YouTube performs the function of passing the received elements in series from the player's memory to its playback facility, resulting in the observed continuous and properly time-sequenced playback.
[1.5.1] as the received media data elements are played, the media player automatically sending additional requests for subsequent media data elements for storage in the memory of the media player	See 1.2.1, showing the additional requests for subsequent media elements, which are successively stored as received (see 1.3). The succession of requests is observed to occur automatically (no user intervention after the initial user instruction to Play is required). This step results from the YouTube-provided application (JavaScript or app) making successive requests from a segment listing in the manifest.
[1.5.2] as required to maintain about a predetermined number of media data elements in the memory of the media player during playing.	The client-side code provided by YouTube performs the function of maintaining about a predetermined number of media data elements in the memory of the media player during playing. This may also be seen in the following trace showing the volume and timing of request activity, in the example desktop platform captures from June 2020:  As can be seen, the player initially makes requests to establish its element store to a desired level, and then periodically makes requests at a rate approximately corresponding to the rate of playback, to maintain the store at the desired level as the media is played out of the store during normal playback. YouTube live captures also reflect a parameter called “rbuf,” which WAG believes is related to the receive buffer of the client. The following trace, taken well into reception of a live stream, reflects the client maintaining an rbuf value in the range of 20,000:

Claim Language

Contention

%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&headm=3&rnm=17&rbuf=0
cver=2.20210202.08.03&headm=3&rnm=18&rbuf=0
cver=2.20210202.08.03&qq=5165042&rnm=19&rbuf=25008
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165042&rnm=20&rbuf=25020
cver=2.20210202.08.03&qq=5165043&rnm=21&rbuf=8946
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165043&rnm=22&rbuf=8688
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165044&rnm=23&rbuf=12695
cver=2.20210202.08.03&qq=5165044&rnm=24&rbuf=11438
cver=2.20210202.08.03&qq=5165045&rnm=25&rbuf=15954
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165045&rnm=26&rbuf=15949
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165046&rnm=27&rbuf=19953
cver=2.20210202.08.03&qq=5165046&rnm=28&rbuf=18196
cver=2.20210202.08.03&qq=5165047&rnm=29&rbuf=22211
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165047&rnm=30&rbuf=20958
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165048&rnm=31&rbuf=24964
cver=2.20210202.08.03&qq=5165048&rnm=32&rbuf=24955
cver=2.20210202.08.03&qq=5165049&rnm=33&rbuf=24721
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165049&rnm=34&rbuf=24720
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165050&rnm=35&rbuf=25191
cver=2.20210202.08.03&qq=5165050&rnm=36&rbuf=19189
cver=2.20210202.08.03&qq=5165051&rnm=37&rbuf=23932
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165051&rnm=38&rbuf=19195
cver=2.20210202.08.03&qq=5165052&rnm=39&rbuf=23447
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165052&rnm=40&rbuf=23450
cver=2.20210202.08.03&qq=5165053&rnm=41&rbuf=24462
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165053&rnm=42&rbuf=24455
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165054&rnm=43&rbuf=24211
cver=2.20210202.08.03&qq=5165054&rnm=44&rbuf=20455
cver=2.20210202.08.03&qq=5165055&rnm=45&rbuf=24970
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165055&rnm=47&rbuf=20216
cver=2.20210202.08.03&qq=5165055&rnm=48&rbuf=18970
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165056&rnm=49&rbuf=23471
cver=2.20210202.08.03&qq=5165056&rnm=50&rbuf=19212
cver=2.20210202.08.03&qq=5165057&rnm=51&rbuf=23682
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165057&rnm=52&rbuf=19192
cver=2.20210202.08.03&qq=5165058&rnm=53&rbuf=23197
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165058&rnm=54&rbuf=23197
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165059&rnm=55&rbuf=24452
cver=2.20210202.08.03&qq=5165059&rnm=56&rbuf=24440
cver=2.20210202.08.03&qq=5165060&rnm=57&rbuf=24454
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165060&rnm=58&rbuf=24456
cver=2.20210202.08.03&qq=5165061&rnm=59&rbuf=24969
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165061&rnm=60&rbuf=24961
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165062&rnm=61&rbuf=24967
cver=2.20210202.08.03&qq=5165062&rnm=62&rbuf=19963
cver=2.20210202.08.03&qq=5165063&rnm=63&rbuf=24432
%3D&alr=yes&cprn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&qq=5165063&rnm=64&rbuf=20189
cver=2.20210202.08.03&qq=5165064&rnm=65&rbuf=24966

Overview	Contents	Summary	Chart	N
Name	Value			
expire	1612703510			
ei	tplfYfM6LHCK6AP1sYd4Dg			
ip	107.77.205.129			
id	w_Ma8oQLmSM.5			
itag	135			
altags	133,134,135,136,160			
source	yt_live_broadcast			
requirssl	yes			
mh	wG			
mm	44,26			
mn	sn-n4v75,sn-7a5meknzs			
ms	lva,onr			
mv	3			
mvi	3			
pl	24			
intlcwdbps	273750			
pvpr	1			
live	1			
hang	1			
noclen	1			
mime	video/mp4			
ns	pelCcuN6dX8jn76cq4Wx91V1			
gir	yes			
Headers	Query String	Raw		

HTTP/
Last-Modified: Sun, 07
Content-Type: video/i
Date: Sun, 07
Expires: Sun, 07
Cache-Control: private
Transfer-Encoding: chunki
Alt-Svc: h3=29
X-Walltime-Ms: 161268
X-Bandwidth-Est: 271406
X-Bandwidth-Est2: 967573
X-Bandwidth-Est3: 967573
X-Bandwidth-App-Limited: false
X-Bandwidth-Est-App-Limited: false
X-Bandwidth-Est3: 756014
X-Sequence-Num: 516506
X-Segment-Lmt: 161268
X-Head-Time-Sec: 257631
X-Head-Time-Millis: 257631
X-Head-Sequim: 516506
Access-Control-Allow-Origin: https://
Access-Control-Allow-Credentials: true
Timing-Allow-Origin: https://
Access-Control-Expose-Headers: Client-Vary Origin
Vary Origin
Access-Control-Allow-Origin: https://

Headers Text Hex Raw

Claim Language

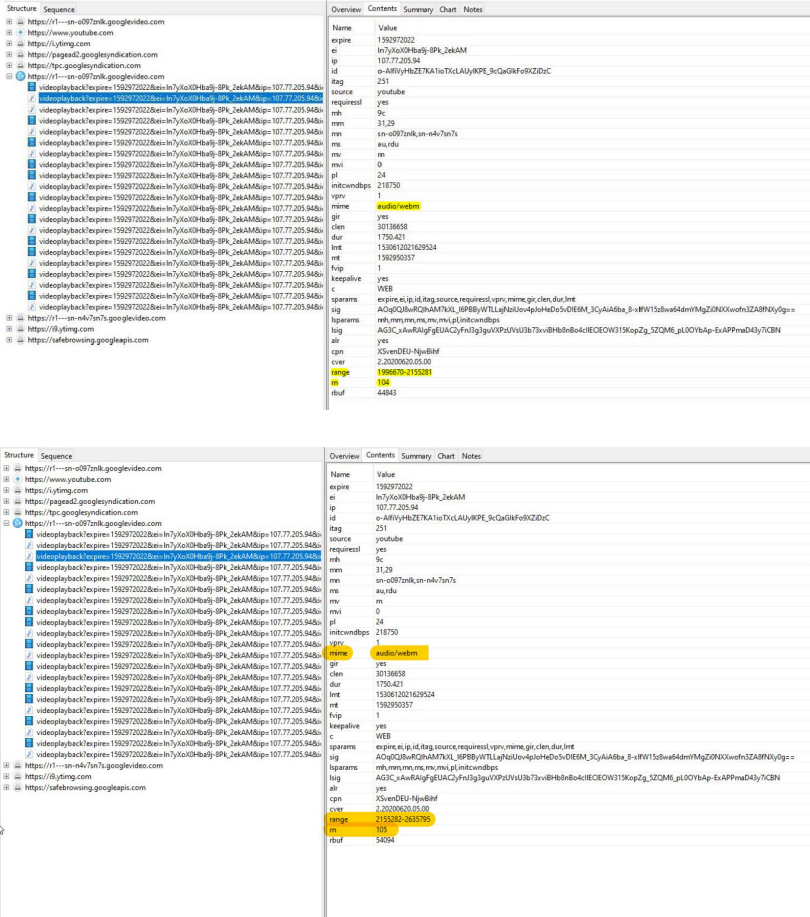
Contention

The screenshot displays a network traffic analysis tool. On the left, a list of HTTP requests is shown, each with a timestamp and a URL. The URL for the selected request is `%3D&alr=yes&cpn=YPO-3LLMyPC-Omgj&cver=2.20210202.08.03&headm=3&rm=17&rbuf=0`. On the right, a detailed view of the selected request is shown, including the request line, headers, and body. The request line is `GET / HTTP/1.1`. The headers include `Last-Modified: Sun, 07`, `Content-Type: video/`, `Date: Sun, 07`, `Expires: Sun, 07`, `Cache-Control: private`, `Transfer-Encoding: chunke`, `Alt-Svc: h3-29=`, `X-Walltime-Ms: 161268`, `X-Bandwidth-Est: 271406`, `X-Bandwidth-Est-Comp: 967573`, `X-Bandwidth-Est2: 967573`, `X-Bandwidth-App-Limited: false`, `X-Bandwidth-Est-App-Limited: false`, `X-Bandwidth-Est3: 756014`, `X-Sequence-Num: 516506`, `X-Segment-Lmt: 161268`, `X-Head-Time-Sec: 257631`, `X-Head-Time-Millis: 257631`, `X-Head-Seqnum: 516506`, `Access-Control-Allow-Origin: https://`, `Access-Control-Allow-Credentials: true`, `Timing-Allow-Origin: https://`, `Access-Control-Expose-Headers: Client-`, and `Vary: Origin`. The body of the request is `Content-Type: video/`.

WAG understands that Apple’s implementation of HLS on the iPhone monitors the receive buffer level and loads Media Segments in advance of when they will be required for uninterrupted playback to compensate for temporary variations in latency and throughput.

WAG has further observed that YouTube MPEG-DASH MPDs have set `minBufferTime=“PT1.5S”`, meaning (according to the MPEG-DASH specification) to buffer at least 1.5 seconds ahead of Presentation Time, *i.e.*, maintain at least 1.5 seconds of data in the buffer, corresponding, at the bandwidth that has been selected by the client, to a predetermined number of media data elements.

This limitation is also met under the doctrine of equivalents. The function of sending additional requests as required to maintain about a predetermined number of media data elements in the memory of the media player during playing is to ensure that the media player does not run out of data, or overflow data, so as to ensure uninterrupted playback of the program, which is evident from the uninterrupted playing of the program on the client device and by the requests discussed above. The way of ensuring that the player memory is not underflowed or overflowed with media data is to monitor the amount of data remaining in the player memory, and/or to monitor available bandwidth between the client and the server, and then balancing anticipated consumption by the client and supply from the server to maintain a level of media data in the player memory that does not

Claim Language	Contention
	underflow or overflow the memory in the media player; this is what happens with the client device, as indicated above by way of both the periodic requests for media data elements and the “rbuf” parameter, both of which show that the client is at least monitoring data remaining in the player memory to ensure there is no underflow or overflow of the memory. The result is that the client obtains a steady stream of media data elements that is sufficient to ensure playback without interruptions, which is evident from the uninterrupted playback on the client device.
2. The method of claim 1, further comprising maintaining in the memory of the media player a record identifying the last media data element received and stored by the media player.	This is reflected, for example, in successive requests, wherein the client is able to recall and increment the identifier of the last media data element received. For example, in June 2020 captured sequence referenced above, the following reflects the requests for rn=104 (and the corresponding range) and the following request for rn=105 (and the corresponding range) (which happen to be for the audio portion of the program):  Similarly, for the video (rn 106 and 108 (and corresponding ranges requested) (the audio and video are being interleaved and rn 107 was audio)):

Claim Language	Contention
video program, from a remote media source via a data connection over the Internet,	tablets, and Google Chromebooks, all of which come with streaming video player apps for streaming from YouTube, preinstalled. Because of encryption of the YouTube streams, and the closed source character of the apps, WAG will seek discovery to verify that, as indicated by external observations, the apps perform in the same infringing manner as the downloaded JavaScript on desktop and iOS devices.
[6.T] the media player comprising	
[6.1] a processor, memory accessible to the processor, and a data connection;	<i>E.g.</i> , Qualcomm Snapdragon processor (phones, tablets), Nvidia Tegra K1 (tablets), Intel Core processors (Chromebooks), 2-16GB RAM, WiFi, LTE connectivity.
[6.2.1] a machine-readable, executable routine containing instructions to cause the processor to send requests from the media player to the media source via the data connection for one or more serially identified media data elements representing an audio or video program, each media data element comprising a digitally encoded portion of the audio or video program and having a playback rate,	<i>E.g.</i> , YouTube app pre-installed on the device, incorporating the same functionality when executed as reported in 1.1.1-1.1.4 and reflected in the evidence recited therein.
[6.2.2] each request specifying one or more serial identifiers of the media data elements requested;	See 1.1.2.
[6.3] a machine-readable, executable routine containing instructions to cause the media player to receive each of the requested media data elements via the data connection, wherein the data connection has a data rate more rapid than the playback rate of the media data elements, and each received media data element is received at a rate as fast as the data connection between the media source and the media player allows;	See 1.2.2-1.2.3 and 6.2.1.

Claim Language	Contention
[6.4] a machine-readable, executable routine containing instructions to cause the processor to store the received media data elements in the memory of the media player;	See 1.3 and 6.2.1.
[6.5] a machine-readable, executable routine containing instructions to cause the received media data elements to be played in series from the memory of the media player; and	See 1.4 and 6.2.1.
[6.6] a machine-readable, executable routine containing instructions to cause the processor to, as the received media data elements are played, automatically send additional requests for subsequent media data elements for storage in the memory of the media player as required to maintain about a predetermined number of media data element in the memory of the media player during playing.	See 1.5.1-1.5.2 and 6.2.1.
7. The device of claim 6, further comprising a machine-readable, executable routine containing instructions to cause the processor to maintain in the memory of the media player a record identifying the last media data element received and stored by the media player.	See 6 and 2
8. The device of claim 6, wherein the serial identifiers are sequential.	See 6 and 3
9. The device of claim 6, wherein the receiving is via a reliable transmission protocol.	See 6 and 4

Claim Language	Contention
10. The device of claim 9, wherein the reliable transmission protocol is TCP.	See 9 and 5
[11.P] A computer program product for receiving and playing an audio or video program by a media player having a processor and a memory accessible to the processor, from a remote media source via a data connection over the Internet, the computer program product comprising a non-transitory computer readable storage medium having program instructions embodied therewith,	YouTube makes and uses computer program products as recited in 6.2.1 and Overview above, in the form of JavaScript files stored on its servers for distribution to Internet users of the YouTube Web Sites, and in the form of application software files as pre-loaded in the devices referenced in 6.P, and YouTube has created and stored such computer program products and then provided them to its own as well as third-party app stores including without limitation the YouTube Web Site, as well as Google Play, Apple, and Roku app stores.
[11.T] the program instructions comprising:	
[11.1.1] instructions executable to cause the processor to send requests from the media player to the media source via the data connection for one or more serially identified media data elements representing an audio or video program, each media data element comprising a digitally encoded portion of the audio or video program and having a playback rate,	<i>E.g.</i> , the machine instructions in the JavaScript stored on machine-readable media on YouTube's servers for the YouTube Web Sites, and in YouTube apps pre-installed on machine-readable media the devices identified in 6.P, and that are stored and provided to third-party app stores, incorporating the same functionality when executed as reported in 1.1.1-1.1.4 and reflected in the evidence recited therein.
[11.1.2] each request specifying one or more serial identifiers of the media data elements requested;	See 1.1.2 and 6.2.2.
[11.2] instructions executable to cause the media player to receive each of the requested media data elements via the data connection, wherein the data connection has a data rate more rapid than the playback rate of the media data elements, and each received	See 1.2.2-1.2.3, 6.2.2, and 6.2.3.

Claim Language	Contention
media data element is received at a rate as fast as the data connection between the media source and the media player allows;	
[11.3] instructions executable to cause the processor to store the received media data elements in the memory of the media player;	See 1.3 and 6.2.2.
[11.4] instructions executable to cause the received media data elements to be played in series from the memory of the media player; and	See 1.4.
[11.5] instructions executable to cause the processor to, as the received media data elements are played, automatically send additional requests for subsequent media data elements for storage in the memory of the media player as required to maintain about a predetermined number of media data element in the memory of the media player during playing.	See 1.5.1-1.5.2 and 6.6.
12. The computer program product of claim 11, further comprising instructions executable to cause the processor to maintain in the memory of the media player a record identifying the last media data element received and stored by the media player.	See 11 and 2
13. The computer program product of claim 11, wherein the serial identifiers are sequential.	See 11 and 3

Claim Language	Contention
14. The computer program product of claim 11, wherein the computer program product is provided as a software application for the media player.	<i>E.g.</i> , YouTube apps; see 11.P. Includes pre-installed apps as well as apps provided through a Google Play or third-party app store. Also includes YouTube JavaScript (<i>e.g.</i> , base.js) on recorded media made and used by YouTube (<i>e.g.</i> , server-side media, from which the YouTube server reads the JavaScript file for online distribution), which also constitutes a software application for a media player.
15. The computer program product of claim 11, wherein the computer program product is incorporated in the media player.	Applies to the software/firmware in devices supplied with a YouTube app installed. See 6.P and 11.P.
16. The computer program product of claim 11, wherein the receiving is via a reliable transmission protocol.	See 11 and 4
17. The computer program product of claim 16, wherein the reliable transmission protocol is TCP.	See 16 and 5