

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

META PLATFORMS, INC.,
Petitioner

v.

TIR TECHNOLOGIES LIMITED,
Patent Owner.

IPR2026-00347

U.S. Patent No. 10,484,442

**DECLARATION OF DR. AVIEL D. RUBIN IN SUPPORT OF META
PLATFORMS, INC.'S PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 10,484,442**

TABLE OF CONTENTS

I. INTRODUCTION	1
II. BACKGROUND AND QUALIFICATIONS	1
III. MATERIALS CONSIDERED	6
IV. APPLICABLE LEGAL STANDARD	6
V. TECHNOLOGY OVERVIEW	12
VI. OVERVIEW OF THE '442 PATENT	15
A. Subject Matter Overview	15
B. File History	17
VII. LEVEL OF ORDINARY SKILL	17
VIII. CLAIM CONSTRUCTION	18
IX. OVERVIEW OF THE PRIOR ART	20
A. Mattis	20
B. Hosur	27
C. Mills	31
D. Sackstein	33
X. DETAILED EXPLANATION OF GROUNDS	36
A. Ground 1A: Mattis in view of Hosur, with or without Sackstein	36
1. Motivation to Combine	36
2. Claims 1/11	45
a. 1[pre]/11[pre]	45
b. 11[a]	46
c. 11[b]	47
d. 1[a]/11[c]	47
e. 1[b]/11[d]	52
f. 1[c]/11[e]	54

g.	1[d]/11[f].....	62
h.	1[e]/11[g]	63
i.	1[f]/11[h].....	67
j.	1[g]/11[i].....	69
3.	Claims 2/12	70
a.	2[a]/12[a]	70
b.	2[b]/12[b]	72
4.	Claim 3/13.....	73
5.	Claims 4/14	74
6.	Claims 5/15	75
7.	Claims 6/16	77
8.	Claims 7/17	78
9.	Claims 8/18	80
10.	Claims 9/19	80
11.	Claims 10/20	82
B.	Ground 2A: Hosur in view of Mattis, with or without Sackstein.....	84
1.	Motivation to Combine	84
2.	Claim 1	89
a.	1[pre]/11[pre].....	89
b.	11[a]	90
c.	11[b].....	90
d.	1[a]/11[c]	90
e.	1[b]/11[d].....	92
f.	1[c]/11[e]	95
g.	1[d]/11[f].....	98
h.	1[e]/11[g]	99
i.	1[f]/11[h].....	101

j. 1[g]/11[i]	103
3. Claims 2/12	104
a. 2[a]/12[a]	104
b. 2[b]/12[b]	105
4. Claim 3/13	106
5. Claims 4/14	108
6. Claims 5/15	109
7. Claims 6/16	111
8. Claims 7/17	112
9. Claims 8/18	114
10. Claims 9/19	115
11. Claims 10/20	117
C. Ground 3A: Mills in view of Mattis, with or without Sackstein.....	118
1. Motivation to Combine	118
2. Claim 1/11	125
a. 1[pre]/11[pre].....	125
b. 11[a]	125
c. 11[b].....	126
d. 1[a]/11[c]	127
e. 1[b]/11[d].....	128
f. 1[c]/11[e]	130
g. 1[d]/11[f].....	132
h. 1[e]/11[g]	134
i. 1[f]/11[h].....	135
j. 1[g]/11[i].....	137
3. Claims 2/12	138
a. 2[a]/12[a]	138

b. 2[b]/12[b] 140

4. Claim 3/13 141

5. Claims 4/14 142

6. Claims 5/15 143

7. Claims 6/16 145

8. Claims 7/17 146

9. Claims 8/18 148

10. Claims 9/19 148

11. Claims 10/20 150

XI. SECONDARY CONSIDERATIONS 151

XII. CONCLUSION.....152

EXHIBIT LIST

No.	Exhibit
1001	U.S. Patent No. 10,484,442
1002	File History of the '442 patent
1003	Declaration of Dr. Aviel D. Rubin
1004	Curriculum Vitae of Dr. Aviel D. Rubin
1005	U.S. Patent No. 6,289,358 (“Mattis”)
1006	U.S. Patent Application Pub. No. 2010/0070608 (“Hosur”)
1007	U.S. Patent Application Pub. No. 2009/0119322 (“Mills”)
1008	U.S. Patent Application Pub. No. 2009/0310668 (“Sackstein”)
1009	U.S. Patent No. 7,860,950 (“Menon”)
1010	Gourley, D., Totty, B., <i>HTTP: The Definitive Guide</i> , O’Reilly, (2002)
1011	Dilley, D., et al., <i>Globally Distributed Content Delivery</i> , IEEE Internet Computing (Sep. Oct. 2002)
1012	Buyya, R., Pathan, M., Vakali, A. (Eds.), <i>Content Delivery Networks</i> , Springer (2008)
1013	Dean, M., et al. (Eds.), <i>Web Information Systems Engineering – WISE 2005 Workshops</i> , Springer (2005)
1014	Plaintiff Tir Techs. Ltd.’s Responsive Claim Construction Brief, <i>Tir Techs. Ltd. v. Amazon.com Services LLC, et al.</i> , No. 25-dv-00865 (W.D. Tex. April 22, 2026)
1015	Schneier, B., <i>Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C</i> , John Wiley & Sons, Inc. (1996)

No.	Exhibit
1016	Ball, M., <i>A Quarter Century of Streaming Video</i> , https://www.thestreamingbook.com/chapters/a-quarter-century-of-streaming-video
1017	Ruether, T., <i>History of Streaming Media [Infographic]</i> , https://www.wowza.com/blog/history-of-streaming-media#h-1990-2000-streaming-debuts-as-a-proof-of-concept (Feb. 21, 2023)
1018	U.S. Patent Application Pub. No. 2002/0065926
1019	U.S. Patent Application Pub. No. 2011/0289533
1020	U.S. Patent Application Pub. No. 2004/0015986
1021	U.S. Patent Application Pub. No. 2004/0117455
1022	U.S. Patent Application Pub. No. 2002/0191696
1023	Shen, B., Lee, S-J., Basu, S., <i>Caching Strategies in Transcoding-enabled Proxy Systems for Streaming Media Distribution Networks</i> , HP Laboratories Palo Alto, HPL-2003-261 (2023)
1024	U.S. Patent Application Pub. No. 2003/0035481
1025	Deneke, T., <i>Scalable Distributed Video Transcoding Architecture</i> , Åbo Akademi Univ., Master of Science Thesis (Sep. 2011)
1026	U.S. Patent Application Pub. No. 2007/0198734
1027	U.S. Patent No. 7,941,453
1028	U.S. Patent Application Pub. No. 2004/0101138
1029	U.S. Patent Application Pub. No. 2007/0204064
1030	U.S. Patent Application Pub. No. 2011/0088076
1031	U.S. Patent Application Pub. No. 2004/0010613

No.	Exhibit
1032	Cacheda, F., Plachouras, V., Ounis, I., <i>A case study of distributed information retrieval architectures to index one terabyte of text</i> , Information Processing & Management, Vol. 41, Issue 5, pp. 1141-1161 (Sep. 2005)
1033	U.S. Patent Application Pub. No. 2007/0250900
1034	Complaint, <i>Tir Technologies Limited v. Meta Platforms, Inc.</i> , No. 8:25-cv-2345, Dkt. 1 (C.D. Cal. Oct. 16, 2025)

LISTING OF CHALLENGED CLAIMS

Reference	Limitation
Claim 1	
1[pre]	A computer-implemented method for streaming video, the method comprising:
1[a]	receiving a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;
1[b]	identifying, based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;
1[c]	generating a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;
1[d]	receiving, from the database, a response indicating whether the key matches a reference key;
1[e]	receiving from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;
1[f]	stitching the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and
1[g]	streaming the stitched optimized version of the original source file for playback on the client computing device.
Claim 2	
2[a]	The method of claim 1, further comprising: storing a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and

Reference	Limitation
2[b]	transmitting, to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.
Claim 3	
3	The method of claim 1, wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.
Claim 4	
4[a]	The method of claim 3, further comprising: transmitting, to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and
4[b]	stitching the different segments of the optimized version of the same original source file with the first portion of the original source file.
Claim 5	
5	The method of claim 1, wherein the key comprises a source content signature, the source content signature being a hash of the original source file.
Claim 6	
6	The method of claim 1, wherein the database comprises data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.

Reference	Limitation
Claim 7	
7[pre]	The method of claim 1, further comprising:
7[a]	determining a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers;
7[b]	transcoding, responsive to the determination, the second segment of the original source file;
7[c]	storing the transcoded second segment in the local cache of the video optimizer; and
7[d]	stitching the transcoded second segment with the stitched optimized version of the original source file.
Claim 8	
8	The method of claim 7, further comprising transmitting to the database, instructions to update the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.
Claim 9	
9[a]	The method of claim 7, wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the second segment, that is stored by one of the plurality of other video optimizers; and
9[b]	transcoding the original source file from a start of the second segment to a start of the third segment.

Reference	Limitation
Claim 10	
10	The method of claim 1, wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.
Claim 11	
11[pre]	A system for streaming optimized video to a client device, comprising:
11[a]	a processor; and
11[b]	a non-transitory computer-readable storage medium storing executable computer program instructions that, when executed by the processor, causes the processor to:
11[c]	receive a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;
11[d]	identify, based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;
11[e]	generate a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;
11[f]	receive, from the database, a response indicating whether the key matches a reference key;
11[g]	receive from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;

Reference	Limitation
11[h]	stitch the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and
11[i]	stream the stitched optimized version of the original source file for playback on the client computing device.
Claim 12	
12[a]	The system of claim 11, further comprising instructions that cause the processor to: store a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and
12[b]	transmit, to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.
Claim 13	
13	The system of claim 11, wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.
Claim 14	
14[a]	The system of claim 13, further comprising instructions that cause the processor to: transmit, to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and
14[b]	stitch the different segments of the optimized version of the same original source file with the first portion of the original source file.
Claim 15	
15	The system of claim 11, wherein the key comprises a source content signature, the source content signature being a hash of the original source file.

Reference	Limitation
Claim 16	
16	The system of claim 11, wherein the database comprises data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.
Claim 17	
17[pre]	The system of claim 11, further comprising instructions that cause the processor to:
17[a]	determine a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers;
17[b]	transcode, responsive to the determination, the second segment of the original source file;
17[c]	store the transcoded second segment in the local cache of the video optimizer; and
17[d]	stitch the transcoded second segment with the stitched optimized version of the original source file.
Claim 18	
18	The system of claim 17, further comprising instructions that cause the processor to transmit to the database, instructions for updating the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.
Claim 19	
19[a]	The system of claim 17, wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the

Reference	Limitation
	second segment, that is stored by one of the plurality of other video optimizers; and
19[b]	transcoding the original source file from a start of the second segment to a start of the third segment.
Claim 20	
20	The system of claim 11, wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.

I. INTRODUCTION

1. My name is Aviel D. Rubin. I have been retained as an expert witness to provide my independent opinion in regard to the matters at issue in inter partes review of U.S. Patent No. 10,484,442 (“the ’422 patent”). I have been retained by the Petitioner in this proceeding.

2. I am being compensated \$1,050 per hour for my time spent working in connection with this case. My compensation is in no way related to the outcome of this litigation. If called as a witness, I would testify as to the statements and opinions contained in this report.

3. I am not a legal expert and offer no opinions on the law. However, I have been informed by counsel of the various legal standards that apply, and I have applied those standards in arriving at my conclusions.

II. BACKGROUND AND QUALIFICATIONS

4. I have more than 30 years of experience in the fields of computer science and applied cryptography. My professional career has been dedicated to issues relating to software, networks, and computer systems more generally.

5. I received a Bachelor of Science in Computer Science with Honors from the University of Michigan in 1989, a Master of Science in Computer Science and Engineering from the University of Michigan in 1991, and a Ph.D. in Computer Science and Engineering from the University of Michigan in 1994. My Ph.D.

dissertation was titled “Nonmonotonic Cryptographic Protocols” and concerned authentication in long-running network operations.

6. I am a Professor Emeritus in Computer Science at Johns Hopkins University (the “University”). I joined the Computer Science faculty at the University in 2003 and was promoted to full professor with tenure in April 2004. As a member of the faculty, I perform research, teach graduate and undergraduate courses in computer science and related subjects, advise undergraduate and Masters students, and supervise the research of Ph.D. candidates and other students. I have taught courses including Networking, Security and Privacy in Computing, Cryptography, and Advanced Topics in Computer Security. In September 2023, I was appointed Professor Emeritus.

7. I also served as the Technical Director of the University’s Information Security Institute (“ISI”). The ISI is the University’s focal point for research and education in information security, assurance, and privacy. The University, through the ISI’s leadership, has been designated as a Center of Academic Excellence in Information Assurance by the National Security Agency and leading experts in the field.

8. I also have significant industry experience. Since 2011, I have served as the founder and chief scientist of Harbor Labs, a software and networking consulting firm specializing in data security and privacy. Previously, from 2005 to

2011, I served as the founder and President of Independent Security Evaluators (“ISE”), a computer security consulting firm. In that capacity, I guided ISE through the qualification process to become an independent testing laboratory for Consumer Union, which produces the Consumer Reports magazine. ISE provided consulting services to various companies on potential security flaws and weaknesses in products and systems. These companies included Fortify Software, Verdasys, and Security First Corp. As an independent testing laboratory for Consumer Union, I managed an annual project where we tested popular antivirus products. Our results were published in Consumer Reports each year for three consecutive years.

9. In addition, I came to the University from AT&T Labs, Secure Systems Research Department, where I spent six years focused on Internet and computer security. Prior to AT&T Labs, I spent two years at Bellcore in its Cryptography and Network Security Research group, also focusing on Internet and computer security issues. I also interned at IBM in 1989. During my time there, I worked on the IBM System/360 family of mainframe computer systems.

10. I am the author or co-author of five books regarding information security issues: Brave New Ballot, Random House, 2006; Firewalls and Internet Security (second edition), Addison Wesley, 2003; White-Hat Security Arsenal, Addison Wesley, 2001; Peer-to-Peer, O’Reilly, 2001; and Web Security Sourcebook, John Wiley & Sons, 1997.

11. I have received numerous awards for my work and research in the field of computer security, including:

- In 2020, I received the Distinguished Service Award from the Institute of Electrical and Electronics Engineers (“IEEE”) Computer Society Technical Committee on Security and Privacy;
- In 2010, I received a Fulbright Scholarship to serve as a Visiting Professor at Tel Aviv University in Israel;
- In 2009, I received a Google Research Award for my work on securing medical records on smartphones;
- In 2007, I received the Privacy Enhancing Technologies Award for Outstanding Research for my work on a security analysis of a cryptographically enabled RFID device;
- In 2004, I was named Baltimorean of the Year by Baltimore Magazine for my work in safeguarding the election process;
- In 2004, I received the Electronic Frontiers Foundation Pioneer Award; and
- In 2000, I received the Best Paper Award at the 9th USENIX Security Symposium.

12. I serve, or have served, on several technical and editorial advisory boards. For example, I served on the Editorial and Advisory Board for the

International Journal of Information and Computer Security. I also served on the Editorial Board for the Journal of Privacy Technology. In addition, I have been Associate Editor of the IEEE Security and Privacy Magazine and served as Associate Editor of the Association for Computing Machinery's ("ACM") Transactions on Internet Technology. I also served as Associate Editor of the journal Communications of the ACM, and I was an Advisory Board Member of Springer's Information Security and Cryptography Book Series. I also have served in the past as a member of the Defense Advanced Research Projects Agency's Information Science and Technology Study Group, a member of the Government Infosec Science and Technology Study Group of Malicious Code, a member of the AT&T Intellectual Property Review Team, Associate Editor of the Electronic Commerce Research Journal, Co-editor of the Electronic Newsletter of the IEEE Technical Committee on Security and Privacy, a member of the board of directors of the USENIX Association (the leading academic computing systems society), and a member of the editorial board of the Bellcore Security Update Newsletter.

13. I have spoken on information security and electronic privacy issues at more than 50 seminars and symposia. For example, I presented keynote addresses on the topics "Security of Electronic Voting" to Computer Security 2004 Mexico in Mexico City, Mexico, in May 2004; "Electronic Voting" to the Secure Trusted Systems Consortium 5th Annual Symposium in Washington, D.C., in December

2003; “Security Problems on the Web” to the AT&T EUA Customer Conference in March 2000; and “Security on the Internet” to the AT&T Security Workshop in June 1997. I also presented a talk about hacking devices at the TEDx conference in October 2011 and another TEDx talk on the same topic in September 2015.

14. I am a named inventor on 10 United States patents, including U.S. Patent Nos. 5,638,446 (“Method for the Secure Distribution of Electronic Files in a Distributed Environment”) and 5,809,140 (“Session Key Distribution Using Smart Cards”).

15. A detailed record of my professional qualifications is set forth in the attached Exhibit A, which is my curriculum vitae, including a list of publications, awards, research grants, and professional activities. My curriculum vitae also lists the matters in which I have served as an expert.

III. MATERIALS CONSIDERED

16. In developing my opinions below relating to the ’442 patent, I have considered the ’442 patent, its prosecution history, and any materials cited throughout my declaration.

IV. APPLICABLE LEGAL STANDARD

17. Counsel has informed me of the legal principles that apply in determining whether patent claims are unpatentable for obviousness. Paragraphs in this section beginning with “I understand,” “I have been informed,” or similar

phrases capture my understanding of the law as provided to me by counsel. The other paragraphs are my opinion.

18. I understand that a claim may be unpatentable as obvious under § 103(a) if the subject matter described by the claim as a whole would have been obvious to a hypothetical “person of ordinary skill in the art” (“POSITA”) in view of a prior-art reference, or combination of prior-art references at the time the claimed invention was made.

19. I understand that “the time the claimed invention was made” is often referred to as the priority or critical date.

20. I understand that, before considering the prior art, the claims are first construed to determine their scope. I understand that the claims are read from the perspective of POSITA at the time of the alleged invention. I understand that the words of a claim are generally given their ordinary and customary meaning as understood by a POSITA in light of the entire patent specification in which they appear, as well as the prosecution history of the patent.

21. I understand, however, that the words of a claim may be given a meaning other than their plain and ordinary meaning if: (1) a patentee expressly and unambiguously defined a term in a way that is different from its plain meaning; or (2) a patentee “disclaims” the full scope of a claim term’s plain meaning, for

instance, by making clear and unambiguous statements during prosecution that a claim term does or does not encompass certain features.

22. I understand that, after construing the claims, the teachings of the prior art are considered. I understand that the prior art includes references that were known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the date the invention was allegedly made by the patent holder. For a printed publication to qualify as prior art, I understand that the Petitioner must demonstrate that the publication was disseminated or otherwise sufficiently accessible to the public. However, I also understand that a POSITA is assumed to be aware of all relevant prior art in the field of endeavor covered by the patent-at-issue and all analogous prior art.

23. I understand that although the ultimate question of the obviousness of a claimed invention is a legal determination, obviousness is based on several factual inquiries, including: the scope and content of the prior art, the differences between the claimed subject matter and the prior art, the level of ordinary skill in the art at the time of the invention, and any “objective indicia” or “secondary considerations” of non-obviousness, which must all be considered.

24. I understand that, in determining the scope and content of the prior art, a reference is considered relevant prior art if it falls within the field of the inventor’s endeavor, or if it is reasonably pertinent to the particular problem that the inventor

was trying to solve. A reference is reasonably pertinent if it logically would have commended itself to an inventor's attention in considering his problem.

25. I understand that, to assess the differences between prior art and the claimed subject matter, the claimed invention must be considered as a whole. I understand that this involves showing that a POSITA, confronted by the same problems as the inventor and with no knowledge of the claimed invention, would have identified the elements in the prior art and combined them in the claimed manner.

26. I understand that there are several rationales for combining prior-art references. For instance, I understand that it is considered obvious to:

- combine prior-art elements according to known methods to yield predictable results;
- substitute one known element for another to obtain predictable results (also called a "simple substitution");
- use the prior-art elements in a predictable way according to their established functions;
- apply a known technique to a known device (method or product) ready for improvement to yield predictable results (also called a "known technique");

- choose from a finite number of identified, predictable solutions, with a reasonable expectation of success (also called “obvious to try”);
- or if there is some teaching, suggestion, or motivation in the prior art that would have led a POSITA to modify a prior-art reference or combine prior-art teachings to arrive at the claimed invention (also called “teaching-suggestion-motivation”).

27. I understand, however, that a POSITA does not need to have the same motivations or reasons to combine the prior art as the inventors, and can arrive at the claimed invention for completely different reasons than the inventors.

28. However, I also understand that the prior art cannot be modified or combined using “hindsight.” I understand that hindsight refers to situations in which the patent-at-issue is used as a framework to pick and choose pieces from the prior art to arrive at the claims, or adopting rationales identified by the patent-at-issue that those in the art would not have recognized. Instead, I understand that the obviousness inquiry is evaluated from the perspective of a POSITA as of the critical date of the patent-at-issue, with all the prior art before them, to determine whether the POSITA would have independently arrived at the claimed invention without the benefit of the patent-at-issue for guidance.

29. I understand that “objective indicia” or “secondary considerations” of non-obviousness include whether: (1) there was a long-felt need for the claimed invention; (2) the claimed invention has achieved commercial success; (3) there was copying of the claimed invention by others; (4) there were failed attempts by others to make the alleged invention; (5) there was praise of the invention in the field; and (6) there was skepticism the claimed invention could be achieved, among a few others.

30. As of the time that I prepared this declaration, the Patent Owner had not identified any secondary considerations of non-obviousness. Therefore, I reserve the right to elaborate on my understanding of these secondary considerations and to address any secondary considerations that Patent Owner may subsequently identify.

31. Finally, I understand that, in an inter partes review proceeding, the obviousness of a claim must be demonstrated by “a preponderance of the evidence,” and that this burden falls on the Petitioner. I understand that the preponderance of the evidence standard requires that a reasonable factfinder could find a material fact is more probable than the nonexistence of that fact. It does not allow for speculation regarding specific facts and is instead focused on whether the evidence more likely than not demonstrates the existence or non-existence of specific material facts. I understand that “preponderance of the evidence” is a lower standard than “clear and convincing evidence” (which requires a fact to be substantially more likely to be true

than untrue), or “beyond a reasonable doubt” (which is an exceedingly high standard that I understand is generally reserved for criminal matters).

32. I have applied the “preponderance of the evidence” standard throughout my declaration.

V. TECHNOLOGY OVERVIEW

33. Accessing information over the internet generally utilizes a request-response model. A client device requests access to a resource (such as a webpage, a database, or multimedia content) stored on another network element (such as another computer or server). Ex-1010, 4-5. The network element that receives the request then determines whether it has the requested resource, and provides a response to the requesting client. *Id.*

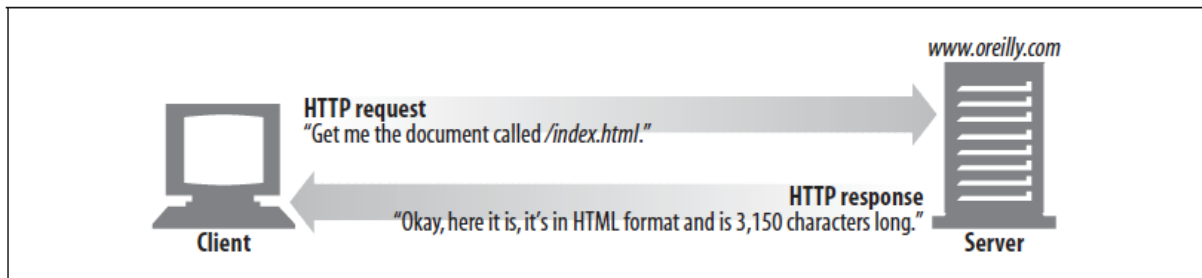


Figure 1-1. Web clients and servers

Ex-1010, Fig. 1-1¹

¹ Images annotated unless otherwise noted.

34. However, “serving Web content from a single location...present[ed] serious problems for site scalability, reliability, and performance,” because “[a]s Web sites become popular, they’re increasingly vulnerable to the flash crowd problem, in which request load overwhelms some aspect of the site’s infrastructure.” Ex-1011, 50. Additionally, “[e]ven if bandwidth isn’t a problem, distance might be,” as more distance between the client device making the request and the network element responding to the request “adds delays.” Ex-1010, 163-64.

35. To solve these problems, companies began building networks that would sit between a resource provider’s server (the server of the website owners, media companies, etc. that stores the original versions of any resources) and the client’s device to distribute resources on behalf of the resource providers faster, while avoiding overloading a resource provider’s own server. Ex-1012, 8-10. By the time of the patent, numerous companies such as Akamai and Speedera offered large, well-known CDNs that resource providers could use to serve their resources to clients over the Internet.

36. The resource provider’s server was typically called an “origin” server, because it served as the originating source of any resources the CDN would then distribute throughout its network. Ex-1012, 5, 37 (“The server where the definitive version of the content resides is called [the] origin server. It is updated by the content provider.”). The resource provider’s origin server would then connect to a CDN,

which provided “*caching and/or replica servers located in different geographical locations to replicate content.*” *Id.*, 5. “A cache server makes copies (i.e. caches) of content at the edge of the network in order to bypass the need of accessing origin server to satisfy every content request.” *Id.*, 37.

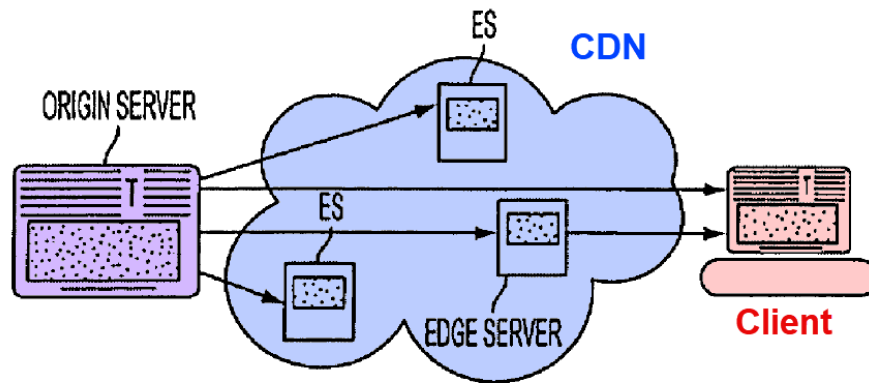


FIG. 1
(PRIOR ART)

Ex-1009, Fig. 1, 3:47-65.

37. Because the Internet is globally accessible, and resources may be accessed from different types of devices, resource providers often provide different versions of the same resource to different clients. *See generally* Ex-1013, 43-52 (describing need to adapt multimedia content stored in a server for compatibility with different devices). CDNs therefore needed ways to keep track of different versions of the same resource, and select which version to provide upon request.

VI. OVERVIEW OF THE '442 PATENT

38. The face of the '442 patent states that it was published on November 19, 2019, and that it was filed on September 15, 2017. However, the '442 patent claims priority to two earlier patent applications. The first is No. 14/846,647, now U.S. Patent 9,800,633, which was filed on September 4, 2015 (which I addressed in a separate declaration in support of *inter partes* review). The second is No. 13/945,747, now U.S. Patent 9,154,361, filed on July 18, 2013, as well as a provisional application (No. 61/673,188) filed on July 18, 2012.

39. I have been informed for purposes of this declaration to assume that the '442 patent is entitled to a priority date of July 18, 2012, and to evaluate the prior art from the perspective of a POSITA at that time.

A. Subject Matter Overview

40. The '442 patent begins by recognizing the problem that “[m]obile devices, such as smart phones and tablets, have become prevalent in recent years,” and that users of these devices often stream videos “at varying qualities, frame rates, frame sizes and compression levels.” Ex-1001, 1:25-32. However, “[i]n many cases, videos available on the network are not optimized for wireless transmission or viewing on mobile devices and are streamed at a given quality...agnostic to network conditions.” *Id.*, 1:32-35.

41. To solve this problem, the '442 patent describes a network that “optimizes” video delivery to clients by offering different versions of the same video files. *Id.*, 1:43-56. To accomplish this, the patent states that when a client requests a video from the network, the request will include a “key” that identifies the requested video, as well as parameters that may be associated with a specific version of the requested video such as the “bit rate, frame width, frame height, sample rate, audio properties and the like.” *Id.*, 6:45-7:29.

42. When the network server receives the request with the “key,” it first determines whether the specific version of video being requested already exists in one of the network’s servers. *Id.*, 9:11-37, 12:30-54. Specifically, the network takes the “key” in the request and compares it to a database of “reference keys” that represent the different videos (and versions of videos) being stored in the network to determine whether there is a match. *Id.* If the “key” matches a “reference key,” that means the specific version of video being requested is already stored in the network, and that version is then located and served to the requesting client. *Id.* If the “key” does not match any “reference key,” though, that means the video—or specific version of the video—being requested does not already exist in the network, and the parameters associated with the “key” in the request will instead be used to create (or “transcode”) a new version of the requested video that matches those parameters. *Id.*, 9:11-37, 12:30-13:13.

43. However, the '442 patent also recognizes that storing video requires significant amounts of memory, and storing complete copies of multiple versions of the same video is “an inefficient use of storage space.” *Id.*, 8:22-42. To avoid these inefficiencies, the '442 patent explains that “portions” of different versions of the same video may be stored in the CDN. *Id.* That way, when a client requests a specific version of a video, the first portion of that version is already available and may be sent to the client immediately while the network locates or transcodes other portions of the video. *Id.*

B. File History

44. As I previously mentioned, the application that led to the '442 patent was filed on September 28, 2017. Ex-1002, 1, 7.

45. On September 25, 2019, the Patent Office issued a Notice of Allowance, noting that two of the references the Examiner considered to be closest to the patent (Taylor, U.S. App. Pub. No. 2010/0106770, and Koopmans, U.S. App. Pub. No. 2012/0030212) did not teach all of the claimed combinations of features. *Id.*, 171-176. The '442 patent issued after that. *Id.*, 207.

VII. LEVEL OF ORDINARY SKILL

46. I understand that when interpreting the claims and assessing the prior art, I must do so from the perspective of a person of ordinary skill in the art (“POSITA”) as of the relevant priority date. As I stated previously, my

understanding is that the earliest possible priority date of the '442 patent is July 18, 2012.

47. In my opinion, in the 2012 timeframe, a POSITA would have had at least a bachelor's degree in computer science, electrical engineering, or a related field, and at least two years of work or research experience in the field of networking and network-data management. However, work experience can substitute for formal education and vice versa.

48. I was at least a POSITA as of the earliest possible priority date of the '442 patent, and still am.

VIII. CLAIM CONSTRUCTION

49. For purposes of my analysis in this IPR proceeding, I understand that the terms that appear in the claims of the '442 patent should be interpreted according to their plain and ordinary meaning. I understand that the plain and ordinary meaning of a claim term reflects the ordinary meaning that the term would have had to a POSITA at the time of the alleged invention in the context of the technology described in the patent. I also understand that the structure of the claims, the specification and file history also may be used to better construe a claim insofar as the plain meaning of the claims cannot be understood. Moreover, I understand that even treatises and dictionaries may be used to determine the meaning attributed by a POSITA to a claim term at the time of the alleged invention. I have applied the

plain and ordinary meaning to the terms of the Challenged Claims in light of the specification and file history of the '442 patent.

50. I understand that the words of the claims should be interpreted as they would have been interpreted by a POSITA at the time the invention was made (not today). For purposes of my analysis here, I have used July 18, 2012 (i.e., the earliest alleged priority date of the '442 patent) as the date of invention. Without exception, however, my analysis of the proper meanings of the terms recited in the Challenged Claims would remain substantially the same even if the date of invention occurred anywhere in the mid-2000s to mid-2010s.

51. Although I am not offering any opinions as to the construction of any claim terms, I understand that the construction of certain claim terms is disputed in separate litigation brought by the Patent Owner against a third party. Specifically, I understand that the third party has raised several terms for construction, including: (1) several terms involving “optimization server”/“optimizer”/“to optimize” as requiring transcoding servers/transcoding; and (2) “stitch” as “combining into one file”. *See* Ex-1014. I also understand that the court has not yet ruled on any of these disputed claim constructions. Regardless, even if the third party’s constructions were adopted, they would not change my opinions because, as I explain below, the prior art taught the claims even under the third party’s proposed constructions.

IX. OVERVIEW OF THE PRIOR ART

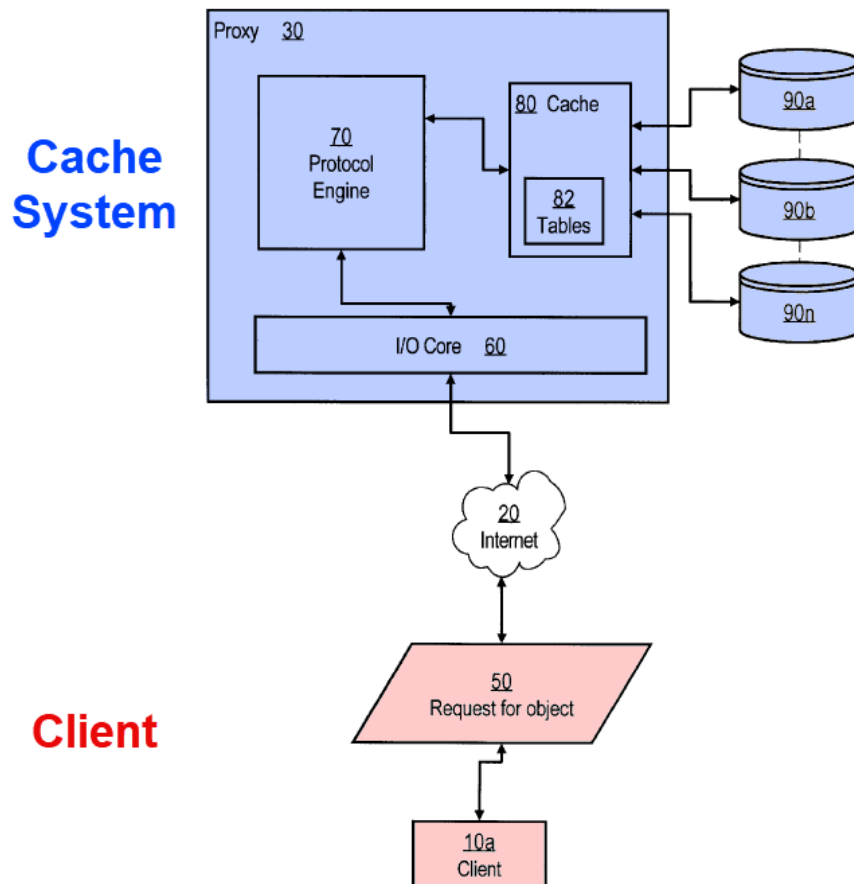
A. Mattis

52. Mattis is a U.S. Patent that was filed on April 15, 1998, and published on September 11, 2001. Mattis, Cover. I therefore understand that Mattis constitutes prior art under pre-AIA 35 U.S.C. §102(b) because it was published (September 11, 2001) more than one year before the earliest possible priority date of the '442 patent (July 18, 2012).

53. Mattis begins by acknowledging that “[a] key factor that limits the performance of the World Wide Web is the speed with which [a] server 40 can supply information to a client via the Internet,” and that speed can be increased by storing “replicas” of content throughout a network. Mattis, 1:53-65. However, “[a] key problem in such caching is the efficient storage, location, and retrieval of objects in the cache.” *Id.*, 2:15-16; *see generally id.*, 1:53-2:52. Moreover, according to Mattis, prior-art systems are unable to “disambiguate and share redundant content that is identical, but has different names,” or “store multiple variants of content with the same name, targeted for particular clients, languages, data types, etc.” *Id.*, 4:3-17; *see also id.*, 5:33-41, 14:16-20, 14:34-36, 14:44-48.

54. To address these (and several other) problems, Mattis discloses systems and methods “for caching and delivering” different versions of “multimedia objects” to clients. Mattis, Abstract, 2:16-18, 4:19-46, 5:30-6:6.

FIG. 2



55. To locate specific versions of files quickly in the cache network, Mattis incorporates “keys” into client requests for a file. Mattis, 8:12-15. Mattis explains that “[t]here are two types of keys, called ‘name keys’ and ‘object keys.’” *Id.*, 8:15-16. An “object key” represents a specific version of specific content. *Id.*, 8:27-45.

56. Mattis explains that “object keys” may be generated by inputting the full contents of a given file through a “hash function,” which outputs a unique short-code representing that file. *Id.*; see also *id.*, 9:58-10:6 (noting use of well-known MD5 hashing algorithms, referencing B. Schneier, “Applied Cryptography”); Ex-

1015, 582-88 (explaining how MD5 hashing algorithms work). If, for instance, a Word Document called petition.docx with its associated metadata, text, and formatting were run through a hashing algorithm, the file's data would be analyzed to output a (preferably) unique 128-bit "key" (a 128-length set of zeros and ones) that is represented using a 32-character hexadecimal string for shorthand.² This "key" would uniquely identify that exact version of petition.docx. But, if another file were also named petition.docx and had as little as one letter changed in its text, or one word bolded, it would produce an entirely different hexadecimal "key," thus indicating the two files are not the same. To demonstrate this, I have run an exemplary sentence through an MD5 hashing algorithm (<https://www.md5hashgenerator.com/>), with just one word spelled differently, to show the vastly different results they produce:

Word	Key (MD5 Hash)
The paper color is white.	5d41402abc4b2a76b9719d911017c592
The paper colour is white.	d08cf6a9484bcde1bab737ee4f8a007d

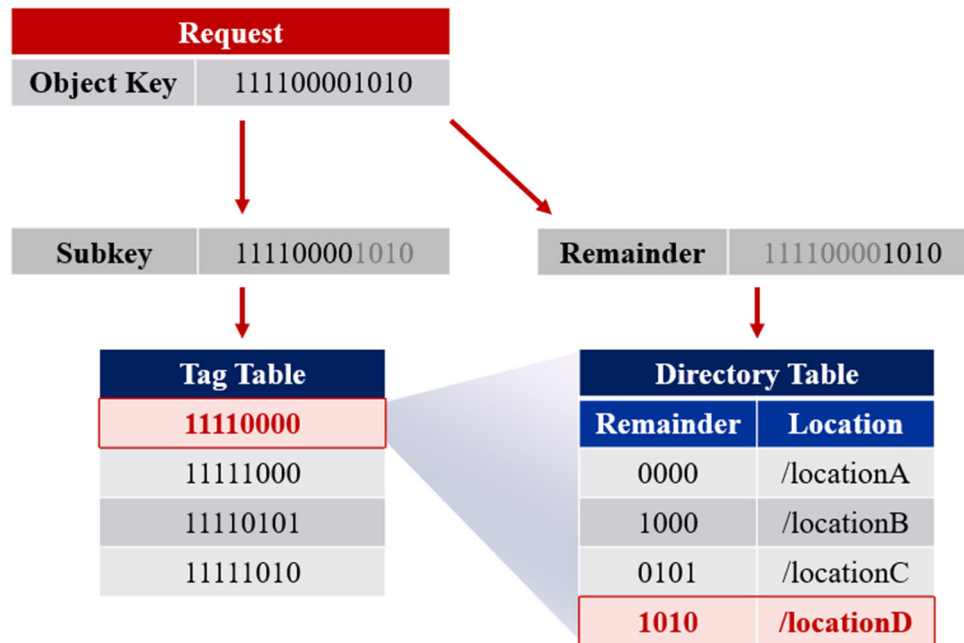
² Because MD5 hashes use 128 bits, there are 2^{128} (or 340 trillion trillion trillion) unique hash-value keys available, meaning the likelihood that any two files have the same hash value is extraordinarily low (but not impossible).

57. Mattis recognizes, however, that clients typically do not know which version of a given file best suits their needs. *Id.*, 10:10-22. For instance, a client may request that a server send a specific picture, but does not know (or care) what format the picture should come in. For this reason, rather than demand that all requests include object keys, Mattis explains that requests may instead include a “name key” representing the name of specific content that may be available in multiple versions. *Id.*, 8:13-26, 10:10-22. Mattis explains that these “name keys” are created by running the ***name*** of the file through a hash function. *Id.*, 8:64-9:4. If, for instance, the file “petition.docx” had different versions written in different languages, every version would have the same “name key” simply because they were titled “petition.docx”. Because a “name key” alone cannot specify a particular version of the file, requests that include name keys will also include additional parameters about the client or client device—such as the client’s “browser..., the version of the browser, the language preferred..., and the type of media content preferred.” Mattis, 14:9-25. These additional parameters are then used to select a specific version of the file to send to the client.

58. When the request and associated key are received by the cache system, the system searches several data tables for a matching key. *Id.*, 11:36-12:49. Two of those tables are a “Tag Table” and a “Directory Table.” *Id.* The reason Mattis provides multiple Tables is for faster searching; each table contains a limited part of

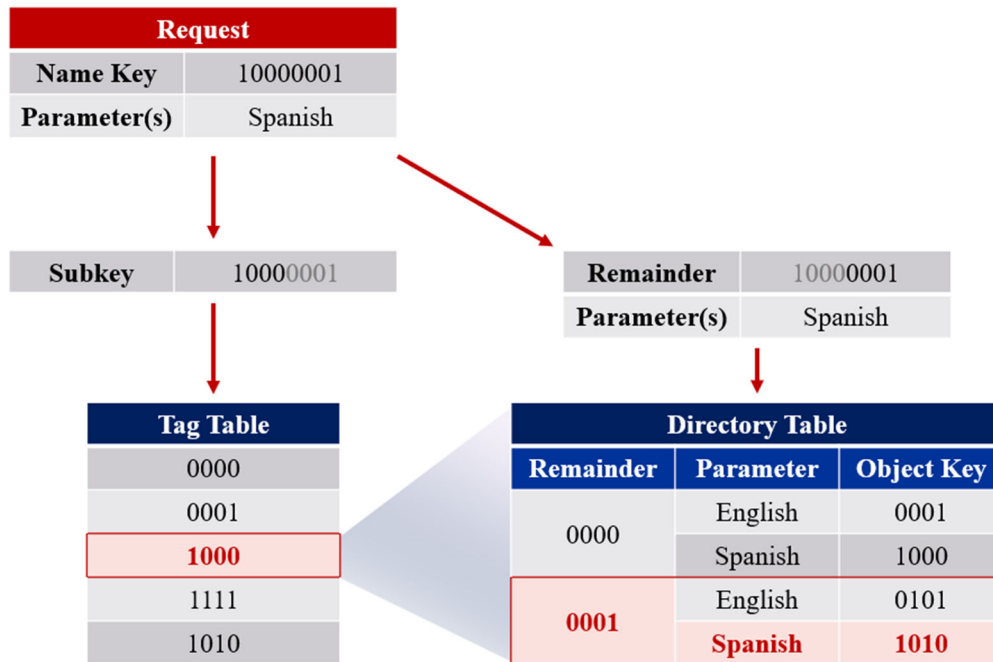
the key, rather than searching the whole key all at once. *Id.*, 9:35-45.³ Specifically, the Tag Table contains a first part of the key, which is searched first to quickly determine whether there is no matching key; if the Tag Table does not have an entry with the first part of the key, that means the cache is not storing the requested file and it need not search for the rest of the key. *Id.*, 9:35-45. But if there is a match in the Tag Table, that means a matching key *possibly* exists, so the rest of the key must be searched in the Directory Table and a matching remainder identifies the location of the content in the cache. *Id.*, 9:46-57, 11:36-12:49. I have provided below a depiction of how an “object key” in a request is searched in both the Tag and Director Tables, where a first part of the key is compared to the Tag Table, and because there is a match, a second part is searched in the Directory Table to ultimately locate where the file is being stored.

³ Mattis also describes an “Open Directory Table”—in addition to the ordinary Directory Table—that allows for faster searching for popular pieces of content that are more likely to be repeatedly requested by multiple clients, as well as control read-write access in the Tag/Directory Tables. Mattis, 16:37-67.



59. Searches are slightly different for object and name keys. Mattis explains that, when searching for object keys, a matching key in the Directory Table identifies the location of the specific version of the content, which I have depicted previously. *Id.*, 11:58-67. For name keys, however, a matching name key in the Directory Table does not identify the location of the file, but rather: (1) every version of the file that has the same name represented by the name key (which Mattis calls a “Vector of Alternates”), (2) parameters associated with each version of the file, and (3) the object key for each version. *Id.*, 11:67-12:4. After locating the different versions of a file with the same name, Mattis’s system then uses the parameters in the request (e.g., the browser, browser version, language, etc.) to select the version that best matches those parameters to obtain the object key for that specific version. *Id.*, 14:8-

15:51. The object key is then used to locate where that specific version is being stored in the cache system. *Id.* I have depicted this process below:



Ultimately, searches for name and object keys proceed similarly, although searches for name keys involve a few extra steps. In the end, however, Mattis’s system always uses an object key to locate a specific version of a specific file.

60. In addition to locating specific files throughout the cache system, Mattis also describes preferred ways to store these files. Specifically, Mattis explains that “[t]o improve the efficiency of read operations that are used to retrieve objects 50 from the cache 80, certain data aggregation techniques are used when initially storing the data.” *Id.*, 17:3-12. One of the techniques used is to split large pieces of content into smaller pieces for storage, which Mattis calls “fragments.” Mattis,

19:19-20:53. An object key points to the fragments, and fragments of the same object are linked in a way that allows them to be collected and combined when a client makes a request for that object. *Id.*

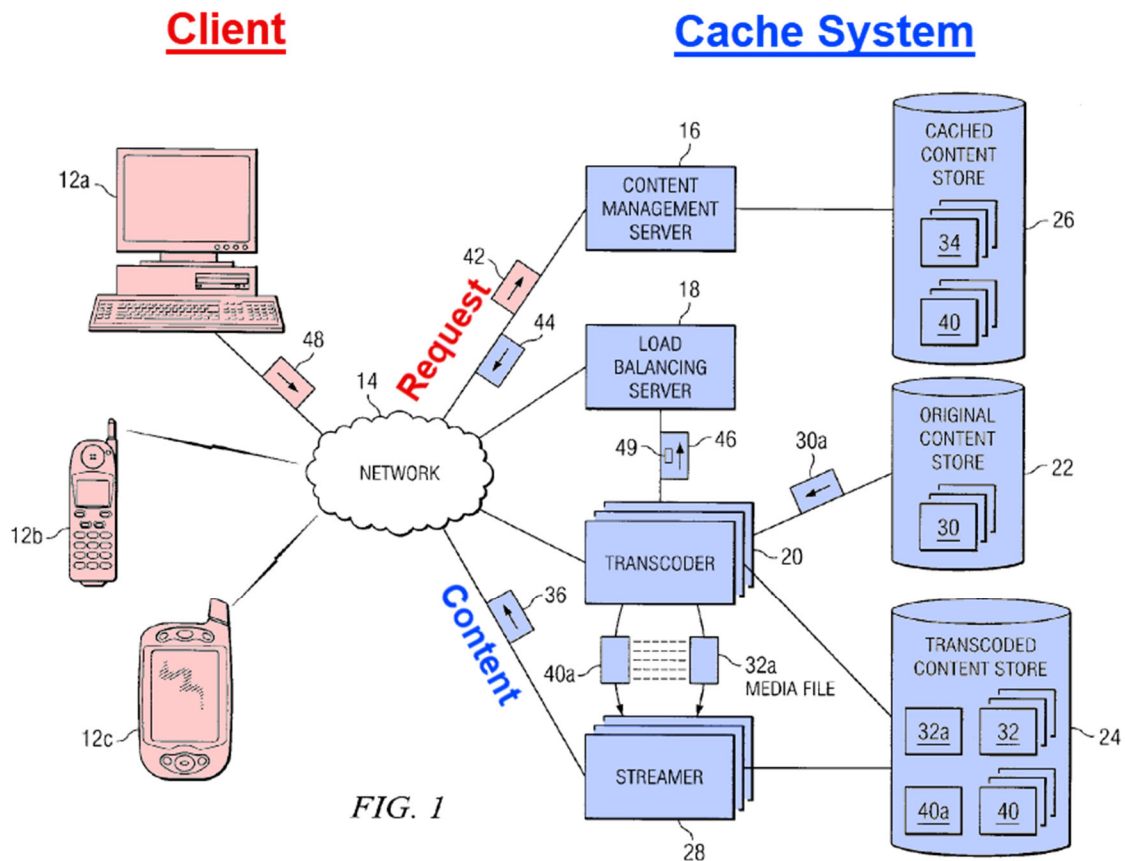
B. Hosur

61. Hosur is a U.S. Patent Application Publication that was filed on September 10, 2008 and published on March 18, 2010. Hosur, Cover. I therefore understand that Hosur constitutes prior art under pre-AIA 35 U.S.C. §102(b) because it was published (March 18, 2010) more than one year before the earliest possible priority date of the '442 patent (July 18, 2012).

62. Hosur begins by explaining that “[t]he variety and availability of audio, video, and multimedia content has increased significantly in the recent past. Internet, mobile Internet, cell phones, personal data assistants, and other mobile media devices fuel constant development and increased demand for media content delivery in a variety of formats.” Hosur, ¶2. However, “[c]onverting content from a source format...to a different target format supported by a particular media player can be time-consuming and computationally-intense.” *Id.*, ¶3.

63. To solve this problem, Hosur describes a system and methods “for delivering media content” by “receiving a request identifying a requested media file” and “generating a modified media file based on the requested media file.” Hosur, Abstract, ¶5. Hosur’s system “includes clients 12a-c, network 14, content

management server (CMS) 16, load balancing server (LBS) 18, one or more transcoders 20,” original-, transcoded-, and cached-content stores (22, 24, 26), and “one or more streamers 28.” *Id.*, ¶14.



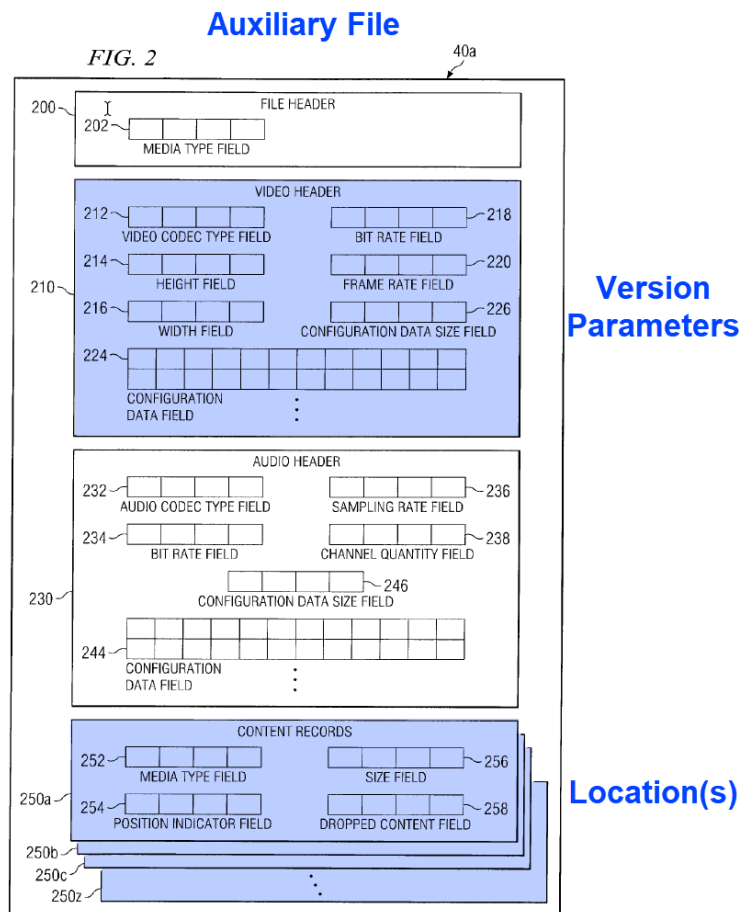
64. Hosur discloses that, first, a client sends a request to CMS 16 for a specific piece of content. *Id.*, ¶¶15, 21, 38. The client’s request “identif[ies] a specific source media file 30” (*id.*, ¶38), and may also specify “the codec, bit-rate, and/or other parameters for the requested content” (*id.*, ¶39). The CMS then “determine[s] an appropriate format (referred to here as the ‘target format’) for the requested content.” *Id.* CMS then determines “whether content from the requested

source media file 30 is already stored in cached content store 26 in the target format,” and if it is, a streamer retrieves the content and sends it to the client. *Id.*, ¶¶40, 95, Fig. 5.

65. If the requested content is not available in the cached-content store, CMS then sends a request to either LBS, or alternatively directly to transcoders, to coordinate transcoding of the requested content into the target format. *Id.*, ¶¶41-42, 96, Fig. 5. The transcoder then retrieves the requested content in a first format, and converts it into the target format. *Id.*, ¶¶43-44, 96. However, rather than wait for the entire file to be transcoded before sending it to the client, Hosur discloses that transcoders can transcode discrete portions of the requested content, and the streamer can send those portions as they become available. *Id.*, ¶¶46, 49, 97. During and after transcoding, the transcoded video (or portion) is stored in a “transcoded content store,” and may then be moved to the “cached content store,” although the cached content store is also optional. *Id.*, ¶¶33-36, 44, 59.

66. To keep track of the different versions (and portions) of content, Hosur discloses that the transcoders generate an “auxiliary file” for each version of content being transcoded. *Id.*, ¶48. “The auxiliary file includes a plurality of content records associated with different portions of the media content in the modified media file and indicating a location of the associated portion.” *Id.*, ¶5. When a transcoder generates a new version of a video—or portion of a video—it writes the location of

that transcoded portion to the auxiliary file, and streamers use the auxiliary file to locate the relevant portion(s) to stream them to the client. *Id.*, ¶¶28, 37, 46-49. The auxiliary file may also include “file header information for transcoded media file 32a,” including “timestamp data, framerate data, resolution data, bitrate data, and any other appropriate information.” *Id.*, ¶46.



67. In sum, Hosur and Mattis are similar in that they both search a network for a specific version of a specific piece of content (which is also stored as discrete portions), but they differ in three primary ways. First, although Mattis discloses

storing and transmitting “multimedia objects” that would be understood to include video (*see, e.g.*, Mattis, 1:37-39, 2:16-18, 5:30-41), Hosur expressly focuses on audio and video multimedia content. Hosur, ¶¶2, 16-17, 18, 32. Second, Mattis stores different versions of already-existing content, but Hosur expressly provides a way to create new versions of content upon request using “transcoders.” Hosur, ¶¶3, 5, 18, 24, 26. And finally, Mattis uses keys and databases to locate specific versions of requested content, but Hosur uses requests and “auxiliary files” that specify both the parameters about each version of content as well as the location(s) of each version in the cache (like Mattis’s name key). Hosur, ¶¶5-6, 27, 46-48.

C. Mills

68. Mills is a U.S. Patent Application Publication that was filed on November 7, 2007 and published on May 7, 2009. Mills, Cover. I therefore understand that Mills constitutes prior art under pre-AIA 35 U.S.C. §102(b) because it was published (May 7, 2009) more than one year before the earliest possible priority date of the ’442 patent (July 18, 2012).

69. Mills is directed to systems and methods “for managing media files having different format characteristics.” Mills, Abstract, ¶1. Mills has many of the same disclosures as Hosur, including “clients 12*a-c*, network 14, content management server (CMS) 16, load balancing server (LBS) 18, one or more transcoders 20,” and original-, transcoded-, and cached-content stores (22, 24, 26)

(*id.*, ¶12), which also operate similarly to Hosur (e.g., transcoding content—including discrete portions—upon request). *Id.*, ¶¶5, 13-15, 20, 21, 26, 28-29, 30, 31, 37-44, Figs. 1, 3A-B.

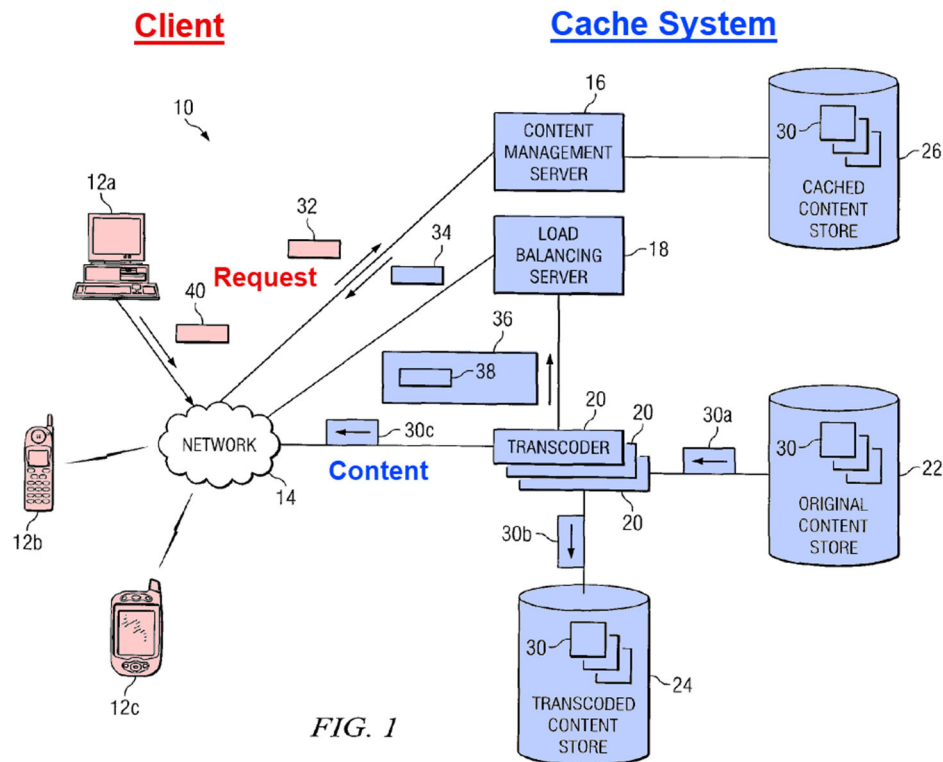


FIG. 1

Mills, Fig. 1 (annotated)

70. Mills and Hosur differ in two primary ways, however. First, Mills does not use “auxiliary files” to locate a specific version of content like Hosur does. Instead, Mills explains that requests may “specify[] the codec, bit-rate, and/or other parameters for the requested media file 30,” which is then used “to determine, based on the information contained in the...request, the output format and characteristics appropriate for the requested media file 30.” Mills, ¶33. Mills does not explain

exactly how the parameters included in the request are used to locate the correct version of content. Second, whereas Hosur focuses on primarily searching its “cached content stores” for the correct version of content, Mills expressly discloses also searching for portions of video stored in transcoded-content stores, in addition to cached-content stores. Mills, ¶61, Fig. 3A (at 308).

D. Sackstein

71. Sackstein is a U.S. Patent Application Publication that was filed on February 23, 2009 and published on December 17, 2009. Sackstein, Cover. I therefore understand that Sackstein constitutes prior art under pre-AIA 35 U.S.C. §102(b) because it was published (December 17, 2009) more than one year before the earliest possible priority date of the ’442 patent (July 18, 2012).

72. Sackstein is directed to “the field of video on demand services, and more specifically, to bandwidth management for video on demand services.” Sackstein, ¶2. Sackstein recognizes that “the rate with which information may be delivered over any network is limited,” and that “[i]f, at any given time, the total bitrate exceeds the maximum bandwidth of the network, information will be discarded...thereby resulting in visible and/or audible artifacts for at least the duration of the excess.” *Id.*, ¶¶4-5. To avoid these issues, “[a] processing technique, known as ‘transrating’ or ‘transcoding’, can be applied to a compressed digital video

content item during streaming in order to reduce the instantaneous bitrate of the stream that will be delivered over a network.” *Id.*, ¶8.

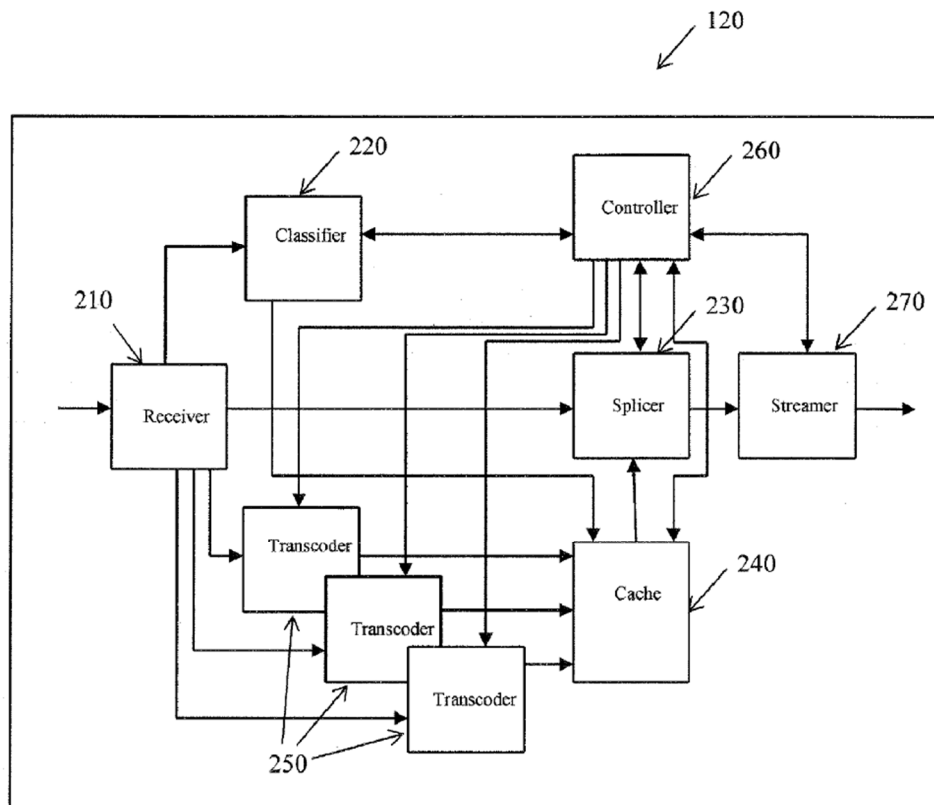
73. Sackstein proposes a system that uses a “classifier 220” to segment video streams, “such that an identical video input would produce identical output segments.” *Id.*, ¶30. For instance, a video can be segmented by a “GOP”—a “group of pictures” (*id.*, ¶32) that are one of the principal ways to group a set of video-images in a predictive-coding scheme. Alternatively, a video can be segmented by scene. *Id.* Each segment is then associated with a “content id” using a hash function and stored in a cache that is indexed using the (hashed) “content id”. *Id.*, ¶¶34, 37.

74. When a particular video is to be streamed to a client from the cache, a controller “monitors the total bitrate of outgoing streams. If the total bitrate is too high, the controller 260 may decide which of the streams will be transcoded and the magnitude of the bitrate reduction.” *Id.*, ¶¶39, 56-57. In other words, Sackstein compresses the video stream so that it consumes less bandwidth during transmission to the client.

75. To accomplish this, “the controller 260 may check the cache 240 for [] transcoded segments with the same content id and bitrate as requested. If a suitable segment is found in cache 240...the controller 260 may retrieve the suitable segment and send it to the splicer 230. If not, controller 260 may assign one of transcoders 250 to transcode the segment to the required bitrate, which will transcode the

incoming segment and store the transcoded segment in cache 240.” *Id.*, ¶¶39, 58. Splicer 230 “may replace segments of an incoming video stream with the transcoded copy retrieved from cache 240...or directly from the transcoder 250.” *Id.*, ¶40. Streamer then receives “video streams from splicer 230 and transmits them to their designated reception device.” *Id.*, ¶41.

Fig. 2



Sackstein, Fig. 2

76. Sackstein further discloses that “cache 240 may be implemented as a distributed cache. That is, each system 120 may contain storage for segments and may share segments with other systems.” *Id.*, ¶47. With distributed caches, “in

addition to checking the local cache 240 [for a specific transcoded segment], the controller 260 may also issue requests for the segment to the caches of other neighboring systems.” *Id.* Sackstein also manages cache memory using a “cache management algorithm [that] may discard a less frequently used [or less popular] segment and insert a newly transcoded segment” when cache memory is full. *Id.*, ¶¶45, 59.

X. DETAILED EXPLANATION OF GROUNDS

A. Ground 1A: Mattis in view of Hosur, with or without Sackstein

1. Motivation to Combine

77. In my opinion, a POSITA would have been motivated to combine Mattis, Hosur, and Sackstein because each of these three references are directed to the same field of content-delivery networks that store and deliver different versions of multimedia content to clients, just like the ’442 patent. *See* §§IX.1 (Mattis), IX.2 (Hosur), IX.4 (Sackstein); §VI.A (’442 Overview).

78. Moreover, all three references implicate similar principles, and already share many similar features, but focus on slightly different aspects of these delivery systems. Mattis, for instance, is focused on how different versions of a multimedia are stored and located in a network; Hosur is focused on generating new versions of multimedia (as well as ways to locate different versions of multimedia) to expedite playback time; and Sackstein is focused on generating new versions of multimedia

to account for bandwidth constraints in a network. Because each had overlapping processes with slightly different focuses, their teachings would have been recognized as largely complementary of one another with minimal modification.

79. A POSITA would have been specifically motivated to modify Mattis in a few ways.

80. **First**, in my opinion, a POSITA would have implemented Mattis's existing teachings to include video, generally, but particularly in view of Hosur. Mattis discloses a system intended to deliver "multimedia object[s]." Mattis, 1:37-39, 2:16-18, 5:30-41. Although Mattis mentions "documents and images" as exemplary "multimedia objects" (*see, e.g.*, Mattis, 1:37-39, 2:16-18, 5:30-41), Mattis does not expressly mention "video," presumably because, at the time Mattis was filed in 1998, streaming video over the Internet was not prevalent because of bandwidth constraints. Ex-1016; Ex-1017. As Internet networks improved in the early-to-mid 2000s, however, streaming video became much more prevalent. Ex-1016; Ex-1017.

81. By 2012 (the time of the invention), streaming video had become just another conventional form of "multimedia object" to deliver over the Internet. *See, e.g.*, Ex-1018, ¶2 ("Many uses have arisen for distributed communication networks and systems, including viewing multimedia data such as documents, images, audio, and video from remote locations."); Ex-1019, ¶64 (describing content delivery in

which “[t]he content 731 can include various forms of digital content including video or multimedia, e.g., MPEG, still images, e.g., JPEG or TIFF, audio, e.g., MP3, spoken audio, digital documents, executable code, and the like.”); Ex-1020, ¶42 (“[A]n exemplary content/service handling and delivery network 10 includes one or more source systems 12 linked via communication mechanisms 14 to a content/service handling and delivery system 16.... [C]ontent may be audio/video and associated data, on-line or electronic documents, web pages, images, graphics, software programs and scripts, non-linear and multimedia compositions, voice, video, data calls, data messages, and so forth. Content, in its simplest form, is high production value audio/video in a variety of formats, MPEG-2, MPEG-4, Quicktime, Real, ASF, and so forth, delivered with rudimentary interactivity, pause, fast forward, rewind, bookmark, and so forth.”).

82. Based on the general adoption of transmitting video over the Internet, in my opinion, a POSITA would have recognized that Mattis’s teachings would obviously extend to video. Hosur—which was filed a decade after Mattis in 2008—confirms that, as Mattis and Hosur are both directed to delivery of multimedia using similar principles, but Hosur expressly focuses on audio and video content, unlike Mattis’s focus on documents and images.

83. In my opinion, a POSITA would have had a reasonable expectation of success in implementing Mattis to serve video as part of its “multimedia objects”

because Mattis expressly envisioned storing, retrieving, and delivering “any discrete element of information that is delivered from a server.” Mattis, 7:23-27. As a POSITA would have understood (and Hosur confirms), video was one type of “discrete element...delivered from a server.” Additionally, storing, retrieving, and delivering video data is no different than storing, retrieving, and delivering other types of media data that Mattis discloses. The primary difference is that video tends to be larger than other types of multimedia objects, but Mattis also envisions storing objects that are large. Mattis, 19:25-38.

84. **Second**, when serving video through Mattis’s network, in my opinion, a POSITA would have also been motivated to incorporate video transcoders into Mattis’s system, as taught by Hosur, to generate new versions of video upon request. A POSITA would have been aware that the caches in content-delivery networks have limited cache memory, and video takes up significantly more memory than other forms of multimedia. *See, e.g.*, Ex-1021, ¶7 (“At the server, large files such as video content require large storage areas (typically disk drives) and use many server cycles and disk cycles to retrieve the video from storage and send it on its way out over the network”); *see also* Sackstein, ¶59 (describing memory management for video segments in memory-limited systems). For instance, documents are typically a few kilobytes, images may range from several kilobytes to several megabytes depending on their size, a typical song requires several megabytes, but a feature-length video

can be several-hundred megabytes to several gigabytes, depending on its size. *See, e.g., Menon*, 4:4-10 (“the typical web pages that are handled are small, again in the kilobyte range.... By comparison, a 1 hour video that represents information at a rate of 4 mega-bits per second (4 Mbps), would require 14,400 Megabits of storage and the corresponding amount of network bandwidth for each edge server”).

85. To avoid the memory burden associated with video, networks that served video would routinely incorporate transcoders—a hardware or software module that converts video from one format to another—rather than pre-generate and store every possible version of a video that may be supported by the network and/or client devices. *Hosur*, ¶7 (noting transcoding discrete portions of video as needed would “decreas[e] the time necessary to deliver the content” and provide for “rapid delivery of media files in particular formats”); *Ex-1022*, ¶44 (“A transcoder 315 operates to reduce the memory and bandwidth requirements of the video”); *Ex-1023*, 2-4 (explaining how transcoding provides tradeoffs in memory and processing requirements). Incorporating transcoders into *Mattis* would “significantly increase content available to media players,” (*Hosur*, ¶7), without requiring prohibitive amounts of memory.

86. In my opinion, a POSITA would have had a reasonable expectation of success in making this modification because *Mattis* already envisioned incorporating “software for dynamically generating...resources in response to requests.” *Mattis*,

1:39-41. Incorporating a video transcoder was a well-known and conventional way to dynamically generate video resources. Hosur, ¶15; Ex-1024, ¶10 (noting a “conventional transcoder”), Fig. 1; Ex-1022, ¶44. Additionally, storing, transcoding, and delivering *portions* of video would have also been straightforward because Mattis already taught storing “fragments” (portions) of large objects (Mattis, 19:19-20:53), and Hosur taught transcoding individual video “portions” (Hosur, ¶¶3, 44-49).

87. **Third**, when explaining the operation of the invention, Hosur primarily describes transcoding and storing portions of a given video in a single transcoder. Hosur, ¶¶42-51. In my opinion, however, Hosur was not limited to transcoding a video using a single transcoder. Hosur explains, for instance, that the network may be provided with multiple transcoders/transcoding modules each with their own local “transcoded-content stores” (Hosur, ¶¶15, 31, 34), that may operate “concurrently so that multiple sets of content can be modified simultaneously” (Hosur, ¶78), and enable “realtime transcoding in a high-traffic system” (*id.*, ¶82). Hosur also envisions transcoded-content stores storing “partial file[s]” as opposed to just complete files. *Id.*, ¶33. And Hosur also acknowledges the ability of streamers to “synchronize portions” before streaming, and identify missing portions of a transcoded file. *Id.*, ¶55.

88. In my opinion, Hosur envisioned a system capable of having different transcoders transcode different portions of the same video. For instance, transcoding is processor intensive. Depending on the extent of the transcoding, it can take longer to convert the video than to play back the entire video. Hosur's desire to provide "realtime transcoding"—where the client cannot playback portions faster than they are being transcoded—would have therefore counseled towards using multiple transcoders to simultaneously transcode different portions of the same video to ensure transcoding was not a bottleneck for playback. *See* Ex-1025, Abstract, 1-5 (explaining how distributed computing solves the bottleneck problem of transcoding).

89. Separately, it was common for clients to request video but then stop playback before completing the video. In these cases, there are only a few choices: (1) use limited processing power to complete transcoding even though the client no longer demands it for playback; (2) delete the partially transcoded video, and then redo the same transcoding operations when another client requests the same video; or (3) store the already transcoded portions, and when the same client or a different client requests the same transcoded version of video, transcode only those portions that were not previously transcoded. Given Hosur's focus on transcoding and storing portions of transcoded video, a POSITA would have appreciated that Hosur at least contemplated, if not expressly directed, to the third option.

90. For this reasons, in my opinion, the combined teachings of Mattis and Hosur would have provided a system with multiple transcoders that would both simultaneously transcode different portions of the same video for faster-than-real-time transcoding, as well as store partially transcoded videos in response to one request, and resume transcoding using the same or different transcoders upon subsequent requests.

91. However, even if Mattis and Hosur would not be read this way, in my opinion, a POSITA would have been expressly motivated to modify Mattis and Hosur to provide these features based on Sackstein. Sackstein describes a system and method to enable “concurrent delivery of a plurality of video streams comprising one or more transcoders...and a cache memory to save copies of transcoded segments.” Sackstein, Abstract, ¶¶12-13. Specifically, Sackstein “segments ...[an incoming] video stream,” (*id.*, ¶¶30-33) and saves segments in cache memory using hashes of each segment (like Mattis) (*id.*, ¶¶34-35). When a request for video is received, Sackstein first “check[s] the cache 240 for...transcoded segments,” and if found, “retrieve[s] the suitable segment and send[s] it to the splicer 230” to be combined with other segments. *Id.*, ¶¶39-40, 56-58. Otherwise, Sackstein sends the segment to a transcoder for conversion. *Id.* This operation is consistent with Hosur. However, Sackstein expressly notes that multiple transcoders may operate on different segments of the same video concurrently (*id.*, ¶¶22, 28-29), and to

accommodate distributed transcoding, “cache 240 may [also] be implemented as a distributed cache,” and “each system 120 may contain storage for segments and may share segments with other systems.” *Id.*, ¶47. Thus, when searching for stored segments, the system searches the “local cache,” but also “issue[s] requests for the segments to the caches of other or neighboring systems.” *Id.*

92. A POSITA would have recognized that an expressly distributed system provides two benefits: (1) it provides faster transcoding of the complete video (Ex-1025, Abstract, 1-5) because different transcoders can operate on different segments of the same video at the same time; and (2) it allows different transcoders to apply different transcoding parameters to different portions of video to adapt to network conditions (Sackstein, ¶¶39, 49).

93. In my opinion, a POSITA would have had a reasonable expectation of success in making this modification because Mattis and Hosur already provide all the elements needed to enable this functionality: Hosur provided multiple transcoders that would operate “concurrently” on portions of video, and Mattis provided storing and locating fragments/portions of objects in different memory stores. The only apparent change (if any) would be expressly enabling different transcoders to transcode and store different portions of the same video.

2. Claims 1/11

94. Claims 1 and 11 recite many of the same limitations, except claim 1 is directed to a method and claim 11 is directed to a system. I have therefore addressed the claims' limitations together, although I address the specific structure required of claim 11 separately.

a. 1[pre]/11[pre]: A [computer-implemented method/system] for streaming [optimized] video [to a client device], [the method] comprising:⁴

95. To the extent the preamble is limiting, in my opinion, Mattis combined with Hosur teaches or suggests this limitation.

96. For both claims 1 and 11, Mattis discloses “[a] method [(cl. 1)] and apparatus [(cl. 11)] for caching information objects” (Mattis, 7:4-12), and delivering them “customized for the requesting client” (*a system/computer-implemented method for streaming...to a client device*), (*id.*, Abstract). *See also id.*, 1:4-2:14, 7:4-12, 36:39-46 (“the methods described herein are carried out using a general-purpose programmable digital computer system”), 39:12-17 (streaming), Figs. 1-2, 11.⁵

⁴ Brackets in the claim language indicate the differences between the method claims (claims 1-10) and system claims (claims 11-20).

⁵ I have italicized claim language throughout my analysis.

97. Mattis explains that “information objects” are multimedia such as “Web pages or documents, graphic images, files, text files, and objects created by Web application programs...or other elements stored on a server.” *Id.*, 7:15-29, 5:30-41. Each object “may be associated with a plurality of versions,” and “the appropriate version...is selected.” *Id.*, 5:30-57. Although the ’442 patent does not explain exactly what it means to be “optimized,” the ’442 patent appears to refer to “optimized versions” as being specific versions of a file—and particularly a transcoded version of a file. *See, e.g.*, Ex-1001, 3:43-4:27. If “optimized” means a specific version intended to meet the specific demands or preferences of the client, Mattis teaches it to the same extent as the ’442 patent. But if “optimized” means a transcoded version, Mattis combined with Hosur teach this for the reasons I discuss in detail below. §§X.A.2.d (1[a]/11[c]); X.A.1 (motivation).

98. Even though Mattis does not list “video” as a multimedia object, Hosur taught a system like Mattis’s that cached different versions of video, and selected a specific version to stream to the client (*streaming optimized video*). *See, e.g.*, Hosur, ¶¶12, 15, 25-28, 32-33, 39-44, Fig. 4. As I explained previously, in my opinion, a POSITA would have modified Mattis’s system to stream video as well, as taught by Hosur. §X.A.1 (motivation).

b. 11[a]: a processor; and

99. In my opinion, Mattis discloses or suggests this limitation.

100. Mattis discloses “computer system 1100 upon which...the invention may be implemented” that includes “*a processor.*” Mattis, 36:56-37:35

- c. **11[b]: a non-transitory computer-readable storage medium storing executable computer program instructions that when executed by the processor, causes the processor to:**

101. In my opinion, Mattis discloses or suggests this limitation.

102. Mattis discloses “[c]omputer system 1100 also includes a main memory 1106” (*non-transitory computer-readable storage medium*) “for storing information and instructions [(executable computer program instructions)] *to be executed by processor 1104*”). Mattis, 36:56-38:6. For the system claims 11-20, as I explain in further detail below, the instructions Mattis discloses would cause the processor to perform the various functions I also address below, with or without Hosur and/or Sackstein.

- d. **1[a]/11[c]: [receiving/receive] a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;**

103. In my opinion, Mattis, with or without Hosur, teaches or suggests this limitation.

104. Mattis discloses “[a] client 10a directs a request 50 for an object” and “[t]he incoming request arrives at an input/output (I/O) core 60” (*receiv[ing] a request...from a client computing device*). Mattis, 7:19-65; *id.*, 5:30-57, 1:16-22

(explaining that clients may be computers). Mattis explains that the request contains information to select a specific version of the requested object. *Id.*, Abstract, 5:30-57, 9:16-21, 10:10-22.

105. Mattis also explains that each request includes an object key or a name key (*a key*). Mattis, 8:16-17, 10:10-22.

106. Mattis explains that an object key is a “unique fingerprint or compressed representation of the contents of the object,” preferably generated by running the object through a “hash function” to produce the key. Mattis, 8:26-45. In my opinion, Mattis’ “object key” is a *key specifying parameters to optimize the original source file* because it both represents and identifies a specific version of a specific source file created using specific parameters. In fact, the ’442 patent notes that keys may include “a hash of the original source file.” *See, e.g.*, Ex-1001 at cl. 5; *see also* §IX.1 (Mattis Summary) (explaining hash functions).

107. Regardless, a “name key” represents an object’s name, generated by running the object’s *name* through a hash function. *Id.*, 8:64-9:15, 10:10-21. When a request includes a “name” or “name key,” the request also includes parameters about the client or client device—such as “the Web browser used by the client, the version of the browser, the language preferred by the client, and the type of media content preferred by the client” (*id.*, 14:9-25)—that Mattis’s system uses to identify a specific version of the object requested by name, (*id.* 14:26-15:51, 28:39-30:14,

Figs. 3B, 4A, 9A, C, E). Mattis explains that these additional parameters may be included as part of the name key (*key specifying parameters to optimize the original source file*). Mattis, 14:56-65.

108. Although Mattis discloses that in an “alternative embodiment,” “[t]he request message from the client 10a identifies the requested object by its name,” and “the cache 80 converts the name of the object to a key value,” Mattis, 28:38-58, in my opinion, Mattis was not limited to this embodiment and envisioned including keys in requests themselves, as Mattis expressly discloses with object keys. *See, e.g., id.*, 10:10-16 (providing name keys because “requests for objects typically do not identify requested objects using the object keys.”).

109. But even if Mattis were limited to this embodiment, in my opinion, a POSITA would have been motivated to expressly include either object or name keys in Mattis’s requests because this is an immaterial difference in how requests are structured. For instance, when a user clicks a link to initiate an HTTP request, the link may contain a path to the location of the requested object, or may alternatively provide a parameter—e.g., a key—the receiving element may use to locate the requested object. In fact, it was well-known and common practice that objects would be located in a network by including “keys” in requests, where the key was made by hashing the object’s contents (rather than having the network convert information in requests into the keys), just like Mattis. *See, e.g., Ex-1026, ¶15* (“the device sends

hashes of the actual data of the portions (i.e. the actual image data, JavaScripts, StyleSheets, etc.) and the proxy server compares the received and stored data hashes for the portions to determine if the device already has the data for a particular portion”). For these reasons, in my opinion, a POSITA would have understood that keys would be included in Mattis’s requests, or at a minimum would have been motivated to include them with a reasonable expectation of success.

110. In my opinion, by disclosing including object and name keys in requests to locate a specific version of video, Mattis discloses a request “*to optimize*” a file consistent with how the ’442 patent describes a request “*to optimize.*” Ex-1001, 11:15-12:54 (optimizing by “obtain[ing] the optimized version...which may be...retrieved from a local cache”), Fig. 3A.

111. If, however, “*to optimize*” requires *generating* a new (optimized) version of content, this feature would have been obvious in view of Hosur. Specifically, rather than pre-generate and store different versions of content as Mattis suggests, Hosur taught that “[i]n response to receiving content request 42, CMS 16 may...determine an appropriate format (referred to here as the ‘target format’) for the requested content to allow, facilitate, and/or optimize transmission to and playback by the requesting client 12,” and that “content request 42 may indicate the appropriate format that the requested content should have for transmission to and/or playback by the requesting client.” Hosur, ¶39. Hosur

searches for the content in the “target format” in the content stores, but if it is not found, “CMS 16 may initiate transcoding of a source media file,” (*id.*, ¶¶41-42), and “[t]ranscoder 20 then modifies requested source media file 30a according to a target format” (*id.*, ¶44). *See also id.*, ¶¶15-16, 25-28, 32-33, 91-98, Fig. 5.

112. This order of operation is exactly like the '442 patent. Specifically, as shown in below, in the '442 patent, requests are sent from user device (110) (at 302-314) *before* a database search is performed to determine whether the cache already contains an “optimized version” of the requested video (at 316-320), or whether a new version must be transcoded. Ex-1001, 11:34-12:54.

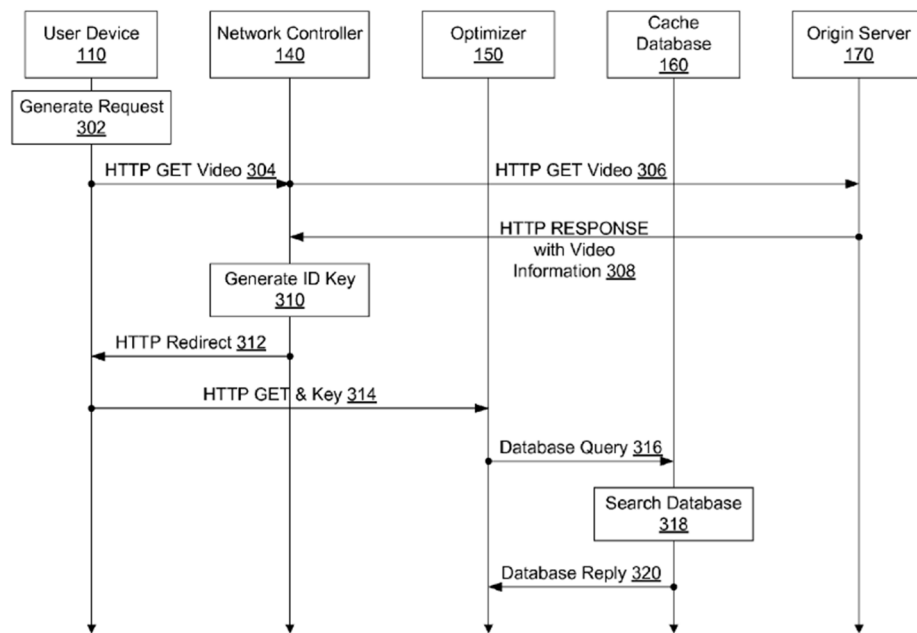


FIG. 3A

113. In my opinion, a POSITA would have incorporated a transcoder like Hosur's into Mattis's system to generate new versions of video upon request for the reasons I provided previously. §X.A.1 (motivation).

- e. **1[b]/11[d]: [identifying/identify], based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;**

114. In my opinion, Mattis combined with Hosur teaches or suggests this limitation.

115. Mattis discloses that, after receiving a request, a specific version of a specific object is located using either the object key or name key and associated parameters (*identifying, based on the key, ...an optimized version of the original source file*). Mattis, 8:6-45 (“Objects on disk are indexed by fixed sized locators, called keys. Keys are used to index into directories that point to the location of objects on disk, and to metadata about the objects”), 10:9-34 (when identified by name, “retrieval of an object in response to a request is a two phase process, where a name key is used to locate the object key, and the object key is used to locate the object itself”), 14:9-15:51 (“When the server 40 receives the HTTP request, it extracts the header information, and selects a variant of the object 52 based upon the values of the header information.”), 28:39-30:14 (exemplary operation of a search), Figs. 3B, 4A, 9A, C, E.

116. Mattis also discloses that each object may be stored as “fragments”—i.e., portions of a complete object. *Id.* at 17:1-18:21. Mattis discloses that the name and/or object key are used to locate fragments—including a first fragment—of a requested version of an object. *Id.* at 19:19-64.

117. Although Mattis does not expressly state that fragments of video are stored in the local cache of a “video optimizer” as claimed, Hosur teaches storing portions of requested video in a local “transcoded content store” (*local cache*) in a video transcoder (*video optimizer*) (Hosur, ¶¶34, 44), and delivering a first portion while other portions are being transcoded (*id.*, 46-49).

118. In my opinion, when implementing Mattis to stream video—as well as incorporating video transcoders to generate new versions of video on demand (§X.A.2.b (1[a]/11[c]))—a POSITA would have understood that individual portions (or fragments) of requested video would be located based on keys (consistent with Mattis), and those portions would be stored by and streamed from the transcoder’s transcoded-content store (consistent with Hosur). §X.A.1 (motivation). That would include using a key to locate a first portion in the transcoder’s transcoded-content store. *Id.*

- f. **1[c]/11[e]: [generating/generate] a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;**

119. In my opinion, Mattis combined with Hosur teaches or suggests this limitation.

120. Mattis discloses a Tag Table and Directory Table that together serve as directories storing object and name keys for reference (*maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files*) to perform searches (*generating a query*). Mattis, 8:12-18 (“Keys are used to index into directories that point to the location of objects on disk”), 8:42-45 (“Since the keys are content-based, and serve as indexes into tables of the cache 80, the cache is referred to as a content-indexed cache. Given a content fingerprint key, the content can easily be found”), 12:9-49 (describing a search).

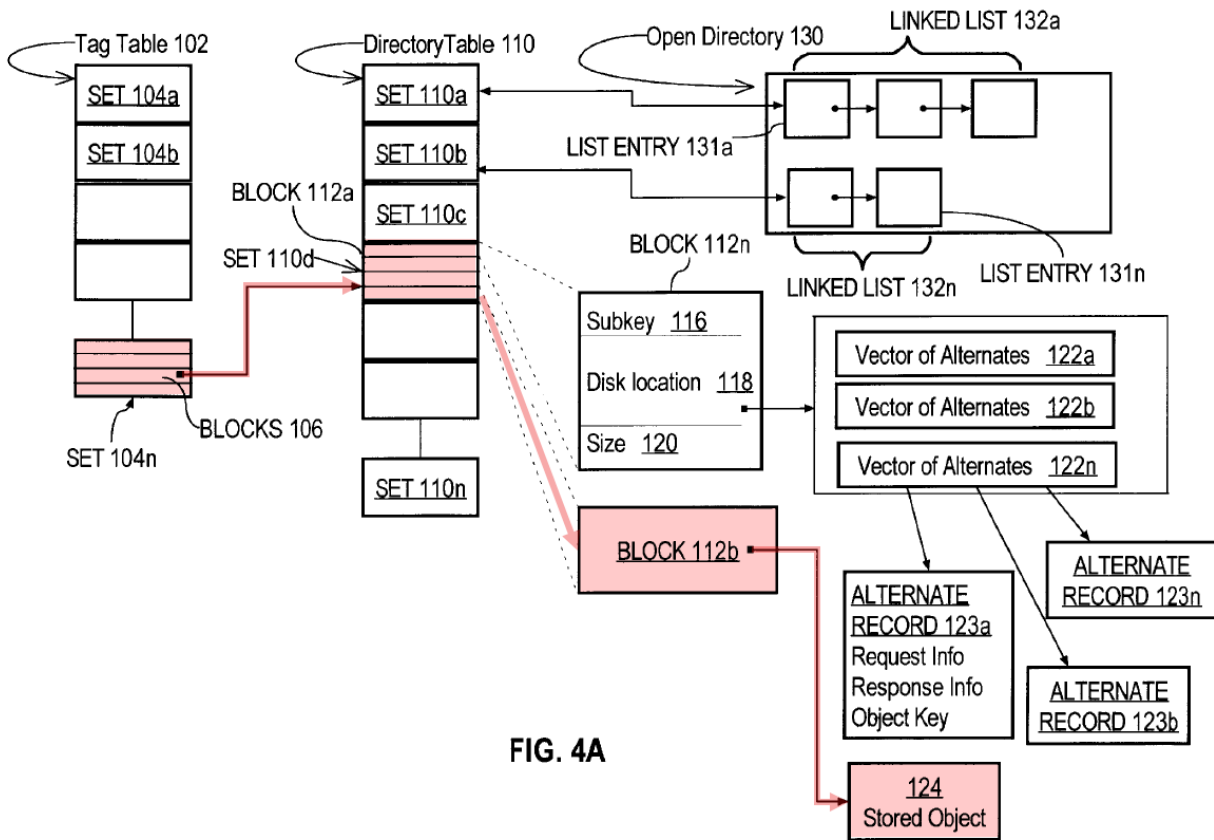
121. In my opinion, Mattis’s Tag and Directory Tables constitute a *database* as claimed, because databases were known to be tables of associative data that used an input (e.g., Mattis’s key) to look up an output (e.g., the location of the requested multimedia object). *See, e.g.,* Ex-1027, 6:55-60 (“The attributes may be used to locate referenced objects, files, and records. One of the attributes may reference a file. If so, the reference can be the file name for the file. For a record from a table in

one of the databases 17 or 19, the reference can include a reference to a database and the column with primary keys for the database table.”); Ex-1028, ¶126 (“The present invention uses the SPI value in order to associate a packet containing encrypted media content with the ECM that is required to generate the key which encrypted that media content. More preferably, client *database 260 contains a table for determining this association*, such that the SPI is at least part of a set of information for identifying the correct ECM for the media content, and hence the correct key which is to be used for decrypting the encrypted media content.” (emphasis added)). More specifically, storing (hashed) keys representing content objects in “databases” was well-known and common practice. *See, e.g.*, Ex-1029, ¶67 (“a different content ID may be generated by applying a predefined hashing or fingerprinting method to each transcoded version of the content and using the resulting hash or fingerprint as the content ID for its associated transcoded version. In this scenario, the content ID of each transcoded version is preferably stored by MMSC 102 in *database 106*” (emphasis added)). And Mattis’s tables serve the exact same purpose as the ’442 patent’s “database”: to “maintains entries corresponding to each optimized video locally cached at a given optimizer.” *Compare* Ex-1001, 2:61-63, *with* Mattis, 8:12-18, 8:42-45.

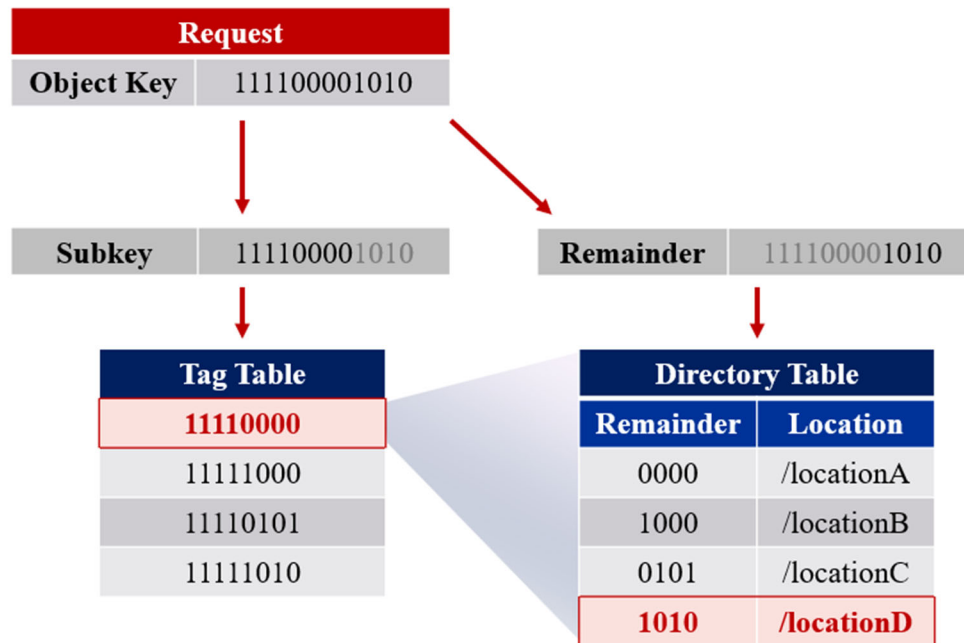
122. Even though Mattis separately notes that the cache should not utilize a database *to store and retrieve objects*, that does not impact my opinion as to whether

the Tag and Directory Tables would be understood to be databases. Specifically, Mattis explains that “[t]he database system approach uses a database system *as a cache*,” but that they “are structured to achieve goals that make them inappropriate for use *as an object cache*.” Mattis, 3:37-59 (emphasis added). In particular, “databases are not naturally good at storing a vast variety *of object* sizes while supporting streaming,” but are “commonly optimized for fixed record sizes” (like Mattis’s keys and location pointers). *Id.* In other words, Mattis suggests that *objects* should not be stored and retrieved in a database, but that does not mean the Tag and Directory Tables—that do not store objects, only small-sized keys—would be implemented as database tables.

123. Mattis discloses the directories are searched using either the object keys or name keys. Mattis, 10:49-53. For object keys, a matching key in the Tag/Directory Tables identifies the location of a specific version of a specific file. Mattis, 11:58-67 (“Directory Table 110 includes a full pointer to a disk location.... If the key was produced based on the content of an object, as described previously, then the disk location value 118 indicates the location of a stored object 124 (or a first fragment thereof), as shown in FIG. 4A”); *see also id.*, 8:40-45 (same), 10:36-11:63 (describing search), 28:39-30:14 (exemplary cache operation), Figs. 4A, 9A, C, E. I have provided below how Mattis depicts the Tag/Directory structures in Figure 4A, which I have annotated to highlight the elements relevant to the object key search:



Because Mattis does not expressly depict in this Figure how the keys are used to traverse this path from the Tag/Directory Tables to the objects, I have also provided an example of how the *values* in an object key would be understood to be searched in the Tag and Directory Tables to locate the object based on the description in the specification:



In my opinion, for the same reasons I consider an “object key” to specify parameters (§X.A.2.b (1[a]/11[c])), the object keys stored in the Tag/Directory Tables also *specify parameters of an optimized version of a source file* because the object key identifies a specific version of a specific source file created using specific parameters. In other words, the same exact video generated with different parameters would produce a different object key.

124. Mattis also explains that objects may be stored in one of a plurality of “storage devices 90a-90n” (Mattis, 8:7-9, 11:53-57) and that object keys in the Tag/Directory Tables points to “a disk location” of the object. *Id.*, 11:22-67. Mattis therefore discloses that, the directories contain *a plurality of reference keys*

corresponding to a plurality of optimized versions of source files stored on a plurality of other...video optimizers.

125. For name keys, Mattis explains that a matching name key in the Tag/Directory Tables identifies all versions of the object (called a “Vector of Alternates”). *Id.*, 11:67-12:8. The “Vector of Alternates...stores a plurality of alternate records...that describe[] an alternative version of the requested object,” including “a particular display and access application” or “a particular parameter or characteristic...such as a human language, a character set, a multimedia data type, or a document encoding scheme” (*reference keys...specifying parameters of an optimized version of a source file*). *Id.*, 14:37-48.

126. Mattis then uses the additional parameters provided in the request—such as “the Web browser used..., the version of the browser used, the language preferred..., and the type of media content preferred” to create a “vector” that points to a version in the alternate records. *Id.*, 14:9-36. A “vector” is a mathematical way to capture the relative values of multiple variables (the variables here being the parameters) into a uniform pointer that enables a search or look-up of a corresponding record in a dataset. Thus, in Mattis’s system, the vector may be formed by weighing, e.g., the browser and user language to locate a single object with matching browser and language compatibility. Once the correct “alternate record” is located, “[t]he alternate records also each store a full object key,” (*id.*,

14:51-55), which specifies the location of each version of the object in one of the plurality of storage devices (*a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other...video optimizers*), (*id.*, 8:7-9, 11:22-67). I have provided below how Mattis depicts the Tag/Directory Table structures in Figure 4A, which I have annotated to highlight the elements relevant to the name key search:

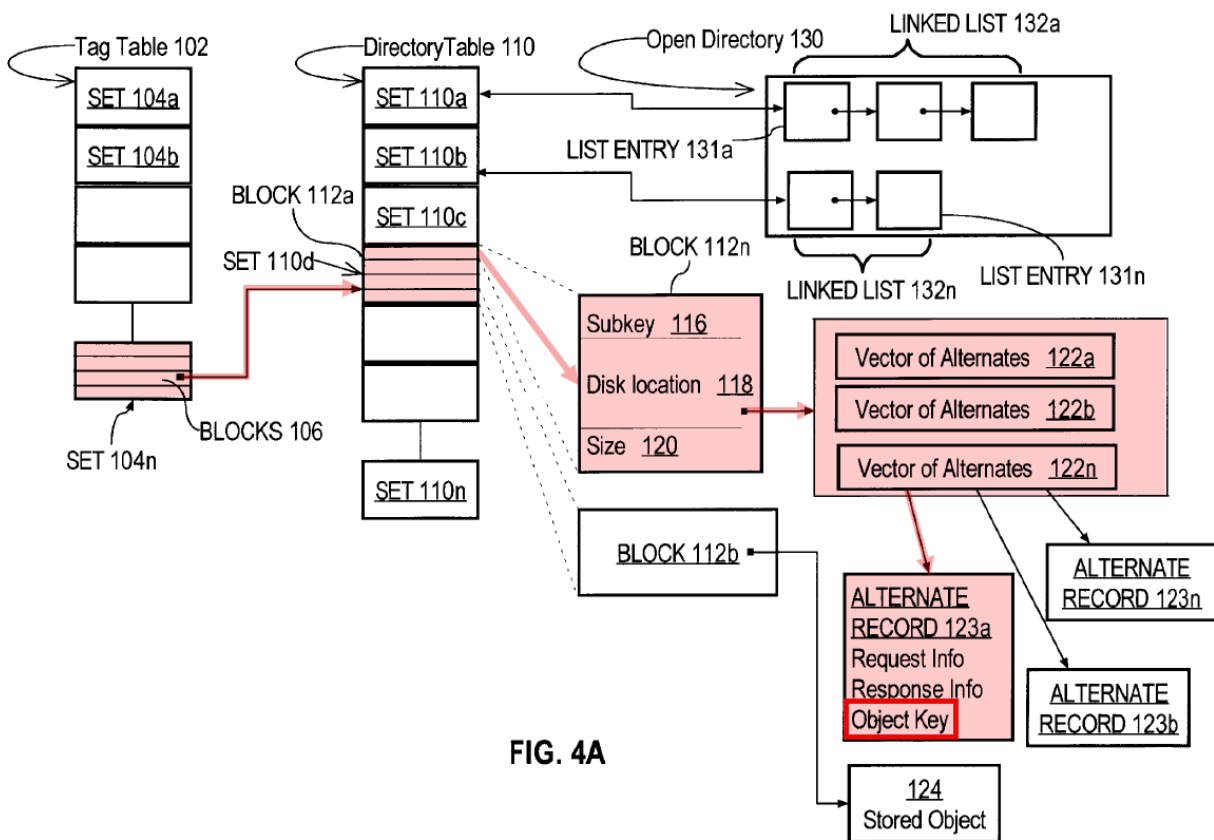
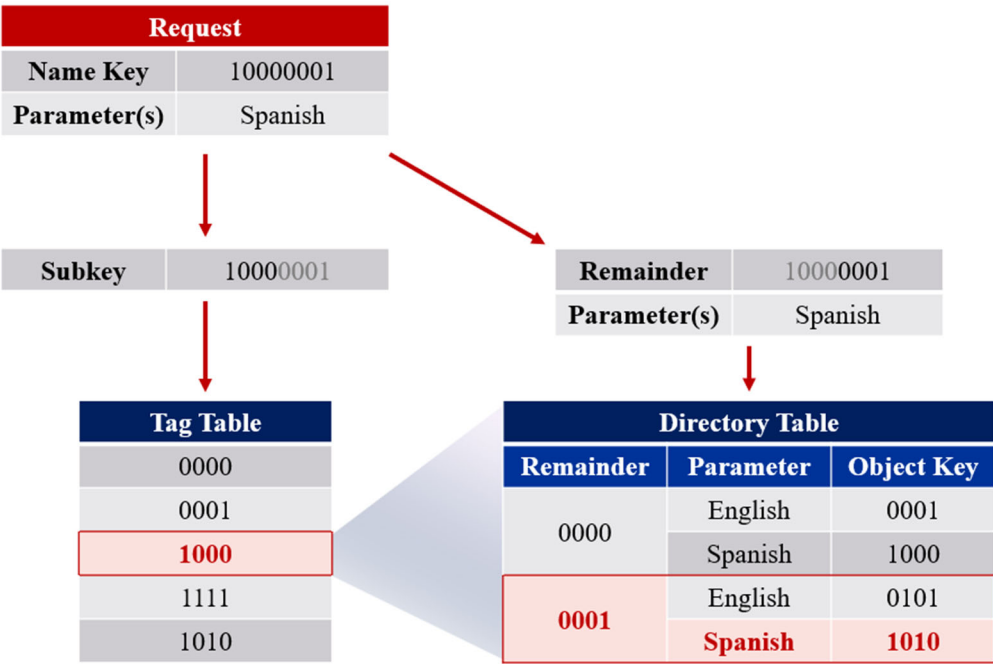


FIG. 4A

Because Mattis does not expressly depict in this Figure how the keys are used to traverse this path from the Tag/Directory Tables to the object keys, I have also provided an example of how the values in a name key and one exemplary parameter

(language) would be understood to be searched in the Tag and Directory Tables to locate the object based on the description in the specification:



In practice, the “Vector of Alternates” can be more complex when weighing multiple variables, however.

127. As I mentioned previously, Mattis does not expressly store different versions of video (or portions of video) in video optimizers. Hosur, however, teaches that “[t]ranscoded content store 24 stores transcoded media files.” Hosur, ¶33; *see also id.*, ¶¶34 (explaining transcoded-content stores may be local to transcoders), 46, 62-64 (retrieving portions for streaming).

128. In my opinion, when implementing Mattis to stream video—as well as incorporating video transcoders to generate new versions of video upon request

(§X.A.2.b (1[a]/11[c])—a POSITA would have understood that Mattis’s Tag/Directory Tables would point to the disk locations of different transcoded-content stores storing different versions of different videos (or portions), consistent with Hosur (*a plurality of optimized versions of source files stored on a plurality of other video optimizers*). §X.A.1 (motivation). In fact, Hosur’s transcoded-content stores would be considered just another storage location like those to which Mattis’s Tables already point. Mattis, 8:7-9, 11:53-57.

g. 1[d]/11[f]: [receiving/receive], from the database, a response indicating whether the key matches a reference key;

129. In my opinion, Mattis discloses or suggests this limitation.

130. Mattis discloses that name and object keys in requests (*the key*) are compared to the entries in the Tag/Directory Tables (*reference keys*) to determine whether there is a match. *See generally* Mattis, 11:36-12:49 (“The search seeks a match of the tag subkey 59 to one [of] the entries. If a match is found...the cache 80 proceeds to seek a hit in the Directory Table”), 9:35-57 (“Misses can be identified quickly by detecting differences in just a small portion of keys”), 10:54-11:4, 11:36-12:49 (describing searches). Specifically, for object keys, the query looks for a direct match, and for name keys, the query looks for a best-matching vector to identify the object key, which is then used to find a direct match to that object key. §X.A.2.f (1[c]/11[e])).

131. Mattis also discloses that, after comparing the keys, the system returns an instruction (such as a bit-flag/update (29:11-40), read/copy (29:55-62, 32:60-66), success/fail indication (30:4-19), error message (33:25-63, 34:25-60), depending on circumstance) to, e.g., the cache and/or a protocol engine depending on whether there is a match (*receiving, from the database, a response indicating whether the key matches a reference key*). See also Mattis, 31:5-18 (describing feedback to, e.g., the protocol engine).

- h. 1[e]/11[g]: [receiving/receive] from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;**

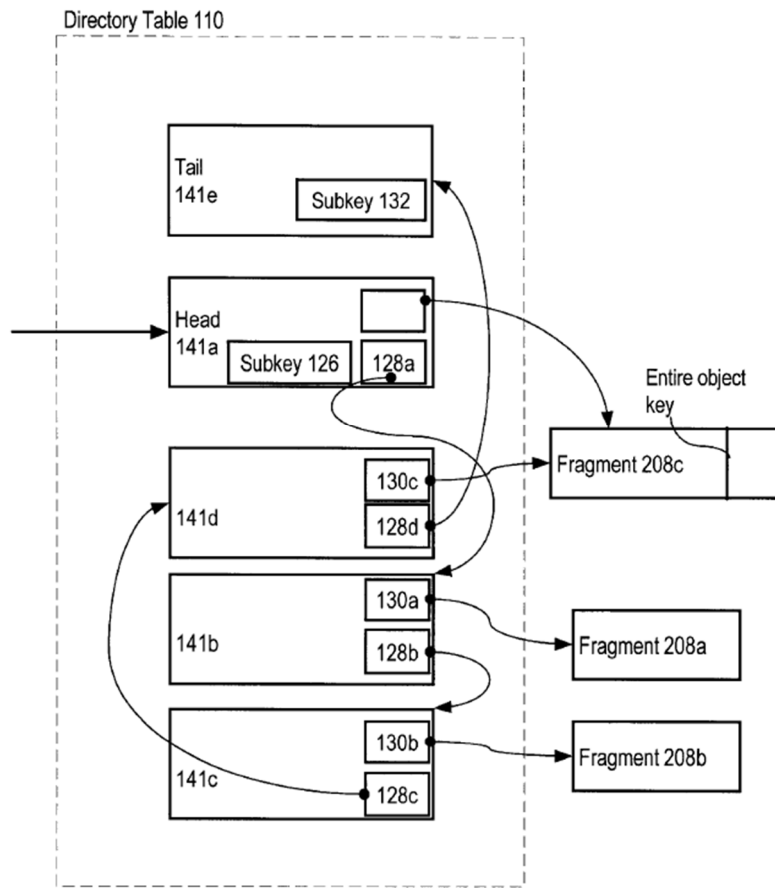
132. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

133. As I explained previously, Mattis discloses that when a request includes an object key, a matching key in the Tag/Directory Tables points to the location of the requested object. *Id.*, 11:58-12:4; §X.A.2.f (1[c]/11[e]). When the request includes a name key, a matching key in the Tag/Directory Tables provides the “alternate records” for different versions of the requested object with their parameters, which are then compared to the request’s parameters to identify the object key—and thus the location—of the best-matching “alternate” version. Mattis, 14:26-36, 15:13-31 (“The cache 80 seeks to match values found in the header

information with request information of one of the alternate records”); §X.A.2.f (1[c]/11[e]). Because Mattis always ends up using the object key to identify the location of the requested object, Mattis discloses *receiving from the database, in response to a reference key matching the key, [an] address...of the optimized version of the original source file.*

134. Mattis also discloses that “[t]he data for an entire stored object...may be stored within multiple fragments.” Mattis, 17:59-66. When an object is stored in separate fragments, a search of the object key in the Directory Table provides a “[b]lock [that] is the head of a chain of blocks that identify successive fragments that contain the object”. *Id.*, 19:39-47.

FIG. 5



In other words, by matching the request's object key to one in the Tag/Directory Tables, Mattis identifies the locations of all the different "fragments" (*segments*) of the requested object.

135. As I mentioned previously, Mattis does not expressly disclose storing different versions of video (or portions of video) in video optimizers. Hosur, however, would have been understood to teach storing and retrieving different portions of a video from transcoded-content stores within different transcoders. §X.A.1 (motivation) (citing Hosur, ¶¶15, 31, 33-34, 55, 78, 82). When combining

Mattis and Hosur, a POSITA would have therefore understood that Mattis's Tag/Directory Tables would identify the address of the transcoded-content stores (*video optimizers*) storing those fragments/portions (*storing a segment*) of the requested version of video, consistent with Hosur (*receiv[ing] from the database, in response to a reference key matching the key, [an] address of another video optimizer storing a segment of the requested optimized version of the original source file*). *Id.*

136. As I also explained previously, however, even if Mattis and Hosur combined is not read this way, Sackstein expressly teaches transcoding and storing different portions of a given video in different transcoders. §X.A.1 (motivation) (citing Sackstein, ¶¶12-13, 22, 28-35, 39-40, 47, 56-58). In my opinion, a POSITA would have been motivated to transcode and store different portions of a video in different transcoders, consistent with Sackstein, to enable faster transcoding as well as dynamic adaptation of individual portions of video. §X.A.1 (motivation). Upon making this additional combination, the *address* received by comparing keys in the request to keys in Mattis's Tables would be the address of *another video optimizer storing a segment of the requested optimized version of the original source file*.

- i. **1[f]/11[h]: [stitching/stitch] the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and**

137. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

138. Mattis discloses that “[t]he object store needs a mechanism to chain fragments together” for retrieval (Mattis, 19:65-66), and “large objects can be efficiently pipelined from the object store” for streaming, (*id.*, 39:15-17). In my opinion, a POSITA would have understood that “pipelining” a set of “chained” fragments would involve retrieving and lining up some number of fragments (e.g., in a memory buffer) to prepare it for streaming. Thus, in instances where more than one fragment is “pipelined” (buffered) together for streaming, that would appear to constitute “stitching” consistent with the ’442 patent. Ex-1001, 17:18-41 (describing “stitching” by aligning key frames prior to streaming).

139. As I discussed previously, Mattis does not expressly disclose chaining fragments of a video from different video optimizers. Hosur, however, would have been understood to store different portions of a transcoded video in different transcoded-content stores. §X.A.1 (motivation) (citing Hosur, ¶¶15, 31, 33-34, 55, 78, 82). And Hosur also discloses that portions are collected into a dynamic “transcoded media file 32a” that “allow[s] streaming of content...while transcoded

media file 32a...[is] still being written.” Hosur, ¶48; *id.*, ¶¶46-47, 53-56. In my opinion, when combining Mattis and Hosur, a POSITA would have therefore understood that a first fragment of a specific version of video (*first portion of the optimized...source file*) is *stitched* with subsequent portions stored in different transcoded-content stores (*stitch[ing]...with the segment of the optimized...source file stored on the other video optimizer*). In my opinion, a POSITA would have either configured the system to buffer multiple chained fragments/portions together in anticipation of streaming like Mattis suggests, or by compiling the fragments/portions into a dynamic “transcoded media file” like Hosur does, as these were two known alternatives that both resulted in combined portions of video for streaming.

140. But even if Mattis and Hosur combined did not teach stitching together portions/segments from different transcoded-content stores, Sackstein expressly describes storing different portions of a video on different transcoders, searching different transcoded caches for transcoded portions of a video, and “splicing” those portions together for streaming to the client. §X.A.1 (motivation) (citing Sackstein, ¶¶12-13, 22, 28-35, 39-40, 47, 56-58). In my opinion, when implementing Mattis to transcode and store different portions of a video on different transcoders consistent with Hosur and/or Sackstein (§X.A.2.h (1[e]/11[g])), a POSITA would have understood that the separately stored portions would be collected from the different

transcoders and chained (like Mattis), combined (like Hosur), or spliced (like Sackstein) together to prepare the portions for streaming to the client. §X.A.1 (motivation).

j. 1[g]/11[i]: [streaming/stream] the stitched optimized version of the original source file for playback on the client computing device.

141. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

142. As I explained previously, Mattis discloses chaining fragments of a specific version of an object together for transmitting it to the requesting client. *See* §X.A.2.i (1[f]/11[h]). Mattis also discloses that “[t]he cache uses streaming I/O, so large objects can be efficiently pipelined from the object store to slow clients, without staging the entire object into memory” (*streaming*). *Id.*, 39:12-17.

143. Mattis does not expressly disclose streaming video for playback on the client device. Hosur and Sackstein, however, both teach streaming portions of video that had been combined or spliced for playback on the client device (*stream[ing] the stitched optimized version for playback on the client computing device*). §X.A.2.i (1[f]/11[h]) (stitching portions of video); Hosur, ¶¶48, 53-54 (incremental streaming of video portions), 57 (“After client 12 begins to receive streamed media content 36 from the selected streamer 28, client 12 may initiate playback of streamed media content 36...while the selected transcoder 20 is still transcoding requested source

media file 30a”); Sackstein, ¶41 (“system 120 may further include a streamer 270 which may receive video streams from splicer 230 and transmits them to their designated reception device”), ¶33 (noting video is for playback). In my opinion, when implementing Mattis’s system to store and stream video consistent with Hosur, a POSITA would have understood that fragments that had been chained, combined, or spliced would be streamed *for playback on the client computing device*, consistent with Hosur or Sackstein, because video is intended to be played back. §X.A.1 (motivation).

3. Claims 2/12

- a. **2[a]/12[a]: The [method/system] of claim [1/11], further comprising [instructions that cause the processor to]: [storing/store] a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and**

144. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

145. Mattis discloses two ways in which a fragment from a different memory may ultimately be stored in the same memory as a first fragment. First, because Mattis discloses pipelining chained fragments for streaming (§X.A.2.i (1[f]/11[h])), a POSITA would have appreciated that the cache handling streaming would (at least temporarily) obtain a copy of the fragment to be streamed. Second, fragments may be reorganized/rewritten into new memory areas for during a “garbage collection”

clean-up process when different objects/fragments are moved around. Mattis, 21:65-22:9, 23:33-47. In other words, fragments that may be stored in different memory areas may subsequently be moved around to co-locate them for efficiency. Additionally, Hosur discloses combining portions into a single “transcoded media file” after transcoding. §X.A.2.h (1[e]/11[g]).

146. In my opinion, when implementing Mattis to transcode and store different fragments or portions of a video on different transcoders, thus resulting in their storage in different transcoded-content stores consistent with Hosur and/or Sackstein (§X.A.2.h (1[e]/11[g])), a POSITA would have understood that the various fragments or portions from different transcoders would be collected in a single transcoder (*storing a segment of the stitched-optimized version...in the cache of the video optimizer*) for streaming, or otherwise efficient access to all fragments when responding to subsequent requests. §X.A.1 (motivation). This would have been accomplished, for instance, by either copying each responsive fragment at the time of access or during a clean-up process consistent with Mattis, or when combined into a single “transcoded media file” consistent with Hosur, as I described previously.

- b. 2[b]/12[b]: [transmitting/transmit], to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.**

147. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

148. Mattis discloses that, when writing, moving, or deleting fragments (*segments*) in memory, “the Directory Table is updated” (*transmitting...instructions to update... in response to storing the segment of the stitched optimized version of the original source file*). Mattis, 18:65-19:5 (writing), 21:38-53 (same), 23:37-52 (moving/deleting), 24:4-15. Hosur likewise teaches updating auxiliary files with the locations of newly transcoded portions of video, and the transcoder issues an instruction to streamer that new portions are available in the transcoded-content store as directed by the auxiliary file. Hosur, ¶¶5-6, 46, 48-49, 53.

149. In my opinion, when combining Mattis and Hosur, with or without Sackstein, a POSITA would have understood that Mattis’s Tag/Directory Tables (*database*) would be instructed to update the Tables to identify the new locations of any fragments/segments moved to the transcoded-content store when rewriting video fragments/segments into a single transcoded-content store. §X.A.3.a (2[a]/12[a]); §X.A.1 (motivation).

4. Claim 3/13: The [method/system] of claim [1/11], wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.

150. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

151. Mattis discloses a matching key provides the locations of fragments of a specific version of an object, and Mattis and Hosur (with or without Sackstein), teach that different portions of a video may be stored in one or more transcoded-content stores. §§X.A.2.g-X.A.2.h (1[d]/11[f]-1[e]/11[g]); §X.A.1 (motivation).

152. In my opinion, when implementing Mattis to transcode and store fragments/portions of a video on different transcoders consistent with Hosur and/or Sackstein (§X.A.2.h (1[e]/11[g])), a POSITA would have understood that a response to a matching key in Mattis's Tag/Directory Tables would indicate that different fragments/portions (*segments*) of a specific version of a video (*optimized version of the same original source file*) are stored in different transcoded-content stores (*a plurality of different video optimizers*). §X.A.1 (motivation).

- 5. Claims 4/14: The [method/system] of claim [3/13], further comprising [instructions that cause the processor to]: [transmitting/transmit], to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and [stitching/stitch] the different segments of the optimized version of the same original source file with the first portion of the original source file.**

153. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

154. Although Mattis describes following a chain to locate fragments (Mattis, 19:19-20:53), Mattis does not expressly describe the particular operation as to how the chain is followed when different memory stores are involved. Hosur similarly teaches that streamers are notified that a transcoder has a portion of a requested video, which streamer accesses, combines with other portions, and streams to the client, (Hosur, ¶¶49, 53-55), but also does not specify exactly how portions are “accessed.”

155. As I explained previously, Mattis combined with Hosur, with or without Sackstein, would have been understood to store different portions of a video in different transcoded-content stores (§§X.A.2.h (1[e]/11[g]), X.A.1 (motivation)). Moreover, network elements were known to access objects by issuing a request for an object and receiving a response with the object. *See, e.g.*, §V (Technology Overview); Ex-1010, Fig. 1-1. In my opinion, when retrieving different

fragments/portions of an object from different transcoders in anticipation of streaming, a network element (like Mattis's cache or Hosur's streamer) would request fragments/segments from different transcoded-content stores, and the transcoders storing the fragments/portions would respond by providing them. This is, for instance, exactly what Sackstein expressly describes: that "the controller 260 may also *issue requests* for the segment to the caches of other or neighboring systems." Sackstein, ¶47 (emphasis added).

6. Claims 5/15: The [method/system] of claim [1/11], wherein the key comprises a source content signature, the source content signature being a hash of the original source file.

156. In my opinion, Mattis discloses or suggests this limitation.

157. Mattis discloses that object keys are a hash of the content of each file, and name keys are a hash of the original source file's name each of which is a *source content signature being a hash of the original source file* (either the file itself, or its name). Mattis, 8:27-50, 8:64-9:15; *see* Ex-1001, 6:64-7:11.

158. If, however, the hash must be of the content of the original source file, this too would have been obvious based on Mattis as a mere design choice as to how video objects should be identified. Specifically, under Mattis's current system, the content of the original source file would already be hashed to provide its object key. Rather than use a name key and parameters to identify a specific version of the original source file, a POSITA would have considered it an obvious alternative to

use the object key of the original source file and parameters. *See, e.g.*, Ex-1029, ¶67 (“For example, the original content ID may be generated by applying a predefined hashing or fingerprinting method to the content and using the resulting hash or fingerprint as the original content ID. The same original content ID associated with the original content received from multimedia content provider 100 may be associated with different transcoded versions of the content as may be prepared for playback by different multimedia devices. Alternatively, a different content ID may be generated by applying a predefined hashing or fingerprinting method to each transcoded version of the content and using the resulting hash or fingerprint as the content ID for its associated transcoded version. In this scenario, the content ID of each transcoded version is preferably stored by MMSC 102 in database 106 together with the original content and/or the original content ID of the original content.”). In my opinion, a POSITA would have been motivated to do so because, as Mattis recognizes, users often do not know the exact version of an object they need. Mattis, 10:10-16. But name keys also have a drawback in that they may be too non-specific to identify a specific object when multiple *different* objects have the same name. This would be avoided by including the object key of the original source file (rather than a name key) and associated parameters in a request to identify a specific version of a specific object.

7. **Claims 6/16: The [method/system] of claim [1/11], wherein the database [comprising/comprises] data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.**

159. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

160. As I explained previously, Mattis discloses the Tag/Directory Tables (*database*) comprises entries (*data entries*) of reference keys associated with the locations of fragments of different versions of a requested object. §§X.A.2.f-X.A.2.h (1[c]/11[e]-1[e]/11[g]). Mattis also discloses updating the entries in the Tag/Directory Tables when fragments are written, moved, or deleted. §X.A.3.b (2[b]/12[b]).

161. Hosur also teaches that the system may store a video in “a select few frequently-requested output formats” (Hosur, ¶60), and transcoding videos using different transcoders, which would therefore be stored in different transcoded-content stores, and updating the auxiliary files to provide their locations (*id.* ¶¶5-6, 46, 48, 53, 59, 78).

162. In my opinion, based on these disclosures, when combining Mattis and Hosur (with or without Sackstein) to store different versions of videos in different transcoded-content stores (§X.A.2.h (1[e]/11[g])), a POSITA would have

understood that the locations in Mattis's Tag/Directory Tables would be those of the transcoded-content stores storing those different versions, consistent with Hosur (*a plurality of optimization servers storing an optimized version of the original source file*). Hosur, Abstract, ¶¶5, 46, 53-54, 70-71, cl. 13, 28; §X.A.1 (motivation).

- 8. Claims 7/17: The [method/system] of claim [1/11], further comprising [instructions that cause the processor to]: [determining/determine] a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers; [transcoding/transcode], responsive to the determination, the second segment of the original source file; [storing/store] the transcoded second segment in the local cache of the video optimizer; and [stitching/stitch] the transcoded second segment with the stitched optimized version of the original source file.**

163. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

164. Mattis does not expressly disclose transcoding files from one format to another. Hosur, however, teaches “determine[ing] whether content from the requested source media file 30 is already stored,” but if not, “initiate transcoding of a source media file 30 containing the requested content.” Hosur, ¶¶40-41. Sackstein similarly teaches that “a cache may be checked for transcoded segments with the same content id and similar or lower bitrates.... If such segments are found, they may be retrieved from the cache and transmitted” but “[i]f suitable segments are not

found in the cache, the segments may be transcoded into their designated bitrate and the transcoded segments transmitted.” Sackstein, ¶58.

165. As I explained previously, when combining Mattis and Hosur, with or without Sackstein, the combined system would have multiple video transcoders capable of transcoding individual video portions upon request (§§X.A.2.b (1[a]/11[c]), X.A.2.h (1[e]/11[g])). A POSITA would have understood that, in operation, some portions of an already-transcoded video would be stored in different transcoded-content stores, but others may be missing. In this case, any missing portions would be transcoded and stitched to generate a complete transcoded video. §X.A.1 (motivation). In fact, others that implemented the same features as Hosur and Sackstein—including portioning multimedia for delivery—performed this exact feature. *See, e.g.*, Ex-1030, ¶¶42 (searching whether fragments already exist), 63 (“Unlike prepared media tracks which have all media fragments to form a complete alternative for the media, the additional alternatives may have none or only a subset of required media fragments prepared and the missing media fragments may be generated through transcoding of existing media fragments prepared for other alternatives when demanded”), Figs. 11a-b.

166. Finally, Hosur discloses that a single transcoder may transcode multiple portions of a video. Hosur, ¶¶44, 46-48. Accordingly, in some instances newly transcoded portions (e.g., *second segments*) would be stored in the same transcoded-

content store as a previously transcoded first portion (*storing the transcoded second segment in the local cache of the video optimizer*).

- 9. Claims 8/18: The [method/system] of claim [7/17], further comprising [instructions that cause the processor to] [transmitting/transmit] to the database, instructions [to update/for updating] the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.**

167. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation for the reasons I have provided previously. §§X.A.3.b (2[b]/12[b]), X.A.8 (cl. 7).

- 10. Claims 9/19: The [method/system] of claim [7/17], wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the second segment, that is stored by one of the plurality of other video optimizers; and transcoding the original source file from a start of the second segment to a start of the third segment.**

168. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

169. As I explained previously, when combining Mattis and Hosur, with or without Sackstein, the combined system would have multiple video transcoders capable of transcoding individual video portions upon request (§§X.A.2.b (1[a]/11[c]), X.A.2.h (1[e]/11[g])). Additionally, both Hosur and Sackstein check

whether transcoded portions/segments are already stored and transcoding missing portions/segments. X.A.8 (cl.7).

170. For these reasons, in my opinion, a POSITA would have understood that, in operation, when a portion of video has not been transcoded (*second segment*) is between two portions that have been (*first portion and third segment*), the portion that has not yet been transcoded would be (*transcoding the original source file from a start of the second segment to a start of the third segment*) to stream a complete transcoded video to the client. §§X.A.2.h (1[e]/11[g]), X.A.8 (cl. 7); X.A.1 (motivation). In fact, others that implemented the same features as Hosur and Sackstein—including portioning multimedia for delivery—performed this exact feature. *See, e.g.*, Ex-1030, ¶¶42 (searching whether fragments already exist), 63 (“Unlike prepared media tracks which have all media fragments to form a complete alternative for the media, the additional alternatives may have none or only a subset of required media fragments prepared and the missing media fragments may be generated through transcoding of existing media fragments prepared for other alternatives when demanded”), Figs. 11a-b.

171. For instance, Hosur acknowledges that, in some instances, a portion of video may be missing or dropped (*second segment*) “between consecutive portions of transcoded media file” (e.g., *first and third segments*), which would either be part of the transcoding operation to reduce the size of the video, or due to a transcoding

error. *See, e.g.,* Hosur, ¶55. Although Hosur discloses that timestamps may be used to allow the streamer/client to cover up these gaps, *id.*, it would have also been apparent that, unless a portion was purposefully dropped, the missing segment may also be re-transcoded and “synchronized” with other transcoded portions.

172. Finally, as I explained previously, because the combined system would provide multiple transcoders that may transcode different portions of the same video, a POSITA would have also understood based on Hosur and/or Sackstein that (some) already-transcoded segments would be stored in different transcoded-content stores (*third segment...that is stored by one of the plurality of other video optimizers*). §§X.A.2.h (1[e]/11[g]), X.A.8 (cl. 7); X.A.1 (motivation).

11. Claims 10/20: The [method/system] of claim [1/11], wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.

173. In my opinion, Mattis combined with Hosur, with or without Sackstein, teaches or suggests this limitation.

174. Mattis discloses a “garbage collection” process that deletes unnecessary objects (or fragments/portions of objects) from the cache based on, e.g., frequency of access (*purges*). Mattis, 21:54-23:52.

175. In my opinion, a POSITA would have expected that a first portions/fragments of a video would be more frequently accessed than later fragments. This is because it was well-known that end users would often start, but

not complete playback of an entire video. Ex-1031, ¶34 (“many people will often start watching a video at its beginning, but will stop watching after a relatively brief period of time.”). Because Mattis implements a garbage-collection process that is fragment-specific and based on frequency of access, then, in many instances later fragments would be deleted (for being less frequently accessed) but the first portion would not. In fact, Sackstein describes a similar deletion process to ensure sufficient cache memory remains available for newly transcoded portions of video, deleting old portions based on frequency of use and/or popularity. Sackstein, ¶¶45, 59.

176. Although Mattis notes that if one fragment of a multi-fragment object is marked for deletion, all fragments are similarly marked for deletion (Mattis, 23:24-31), in my opinion, a POSITA would have understood that Mattis did so because Mattis provided no way to re-create a deleted fragment, meaning deleting one fragment would likely corrupt the entire object. However, when providing transcoders capable of recreating deleted portions/fragments, a POSITA would have appreciated that there would be no need to delete all fragments if some subset are sufficiently active/popular and others are not, because inactive portions would simply be re-transcoded. This is again consistent with Sackstein, which envisioned deleting some, but not all, segments of a video based on popularity of individual segments. Sackstein, ¶¶45, 59.

B. Ground 2A: Hosur in view of Mattis, with or without Sackstein

1. Motivation to Combine

177. In my opinion, not only would it have been obvious to modify Mattis in view of Hosur (and Sackstein), but also to modify Hosur in view of Mattis (and Sackstein). As I explained previously, a POSITA would have been motivated to combine Hosur, Mattis, and Sackstein because each of these three references are directed to the same field of content-delivery networks that store and deliver different versions of multimedia content to clients, just like the '442 patent. *See* §§IX.1 (Mattis), IX.2 (Hosur), IX.4 (Sackstein); §VI.A ('442 Overview). However, the modifications to Hosur would have been different than the modifications to Mattis.

178. Hosur stores and locates different version of video (or portions of video) using “auxiliary files.” Hosur, ¶¶5-6, 46-48, 53-56, 61-64. Hosur explains that each auxiliary file, however, is stored separately with its associated file. Hosur, ¶¶34-35. In other words, it appears as if Hosur’s auxiliary files are distributed throughout the network across different transcoded- and cached-content stores. Hosur, ¶¶34-35. It therefore appears as if Hosur must search different content stores to locate relevant auxiliary files, and then search the auxiliary files to determine whether they are responsive to a request for a specific version of video, or provide some other

mechanism that Hosur does not detail to locate the video files and their associated auxiliary files in the first instance.

179. Rather than use auxiliary files distributed throughout a network to locate relevant portions of requested video, Mattis uses object/name keys and a centralized database of Tag/Directory Tables to locate specific versions of files in a network. *See* §§IX.1 (Mattis Overview), X.A.2.e-X.A.2.f (Ground 1, 1[b]/11[d]-1[c]/11[e]).

180. In my opinion, a POSITA would have been motivated to replace or supplement Hosur's distributed auxiliary files with Mattis's object/name keys and a centralized database of reference keys for several reasons. First, although Hosur details the contents and purpose of the auxiliary file, Hosur does not provide a distributed search algorithm to effectively *locate* relevant auxiliary files, a mechanism to efficiently identify all versions of the same video, or the ability to de-duplicate identical videos. Instead, every version of a video (and possibly every separately stored portion) has its own auxiliary file, so any search algorithm would need to sift through a large number of auxiliary files that are not (expressly) tied to one other to identify both the available versions of a video and then the correct version to be sent to the client.

181. Relative to Hosur, Mattis provides a streamlined approach that uses name and object keys to enable efficient version associates (due to name keys) and

de-duplication of identical objects (due to object keys), thus reducing memory and processing requirements of the system. Mattis, 8:51-9:15, 10:10-22. A POSITA would have understood that querying a centralized directory like Mattis's Tag/Directory Tables would be simpler than searching across different content stores to locate auxiliary files and determine whether they are storing relevant versions of a requested file. Ex-1032. Although there were some known distributed search algorithms that offered some benefits compared to centralized databases, Hosur provides no guidance as to what sort of search algorithm would be used or how the algorithm would be implemented to identify relevant auxiliary files, and thus Mattis provided a straightforward solution to fill the gap in Hosur's disclosure.

182. Regardless, in my opinion, replacing Hosur's distributed auxiliary files with Mattis's keys and centralized database would have amounted to a simple substitution of one known element (distributed auxiliary files) for another (centralized database) to achieve a predictable result (locating portions of a specific version of a specific video). It also would have amounted to replacing one known technique (distributed searching) with another suitable option (centralized searching) to solve a known problem in the art (locating a specific version of a requested video).

183. Alternatively, a POSITA would have been motivated to supplement—rather than replace—Hosur's distributed auxiliary files with Mattis's object/name keys and a centralized database of reference keys. Specifically, Hosur discloses

collecting portions of video in a transcoded media file (Hosur, ¶¶33), and the auxiliary file contains “content records” that “include information regarding the structure and contents of transcoded media file,” and may “include multiple content records 250 each associated with a different portion of the content in transcoded media file.” *Id.*, ¶¶69; *id.*, 46, 53-55. As such, a POSITA would have appreciated that using keys to search a centralized database, as Mattis does, would have provided a straightforward way to locate the transcoded media files and their auxiliary files in the first instance, but the auxiliary files may contain additional information in their content records as to the *contents* of each media file—such as which portions are within each transcoded media file—to assist with streaming those portions and/or transcoding any missing portions.

184. In my opinion, a POSITA would have had a reasonable expectation of success in making either modification (replacement or supplementation) because the content and purpose of Mattis’s object and name keys, as well as Tag/Directory Tables, are extremely similar to Hosur’s auxiliary files (e.g., file parameters and locations); they are both used to locate relevant portions of a specific version of a multimedia object. In fact, Hosur’s use of auxiliary files with associated parameters is nearly identical to Mattis’s use of name keys and associated parameters, but differ as to where the reference parameters are being stored (in auxiliary files versus database tables). The most significant modifications required would be to take the

contents of Hosur's auxiliary files and load them into centralized database tables and have caches search those tables (e.g., using hashed keys), rather than implement a distributed search algorithm as Hosur's disclosure implies but does not describe.

185. Finally, for the reasons I have previously explained (§X.A.1 (Ground 1, motivation)), although Hosur would have been understood as envisioning a system capable of having different transcoders transcode and store different portions of the same video, even if Hosur is not read this way, in my opinion, a POSITA would have been motivated to modify Hosur to expressly provide this feature based on Sackstein. Sackstein describes a process of segmenting a video for transcoding, having a distributed set of transcoders simultaneously transcode the video, storing these portions in their caches, and then locating and splicing the different portions together. (§X.A.1 (Ground 1, motivation) (citing Sackstein, ¶¶12-13, 27-28, 30-33, 39-40, 47, 56-58)). A POSITA would have recognized that this provides two benefits: (1) it provides faster transcoding of the complete video (Ex-1025, Abstract, 1-5); and (2) it allows different transcoders to apply different parameters to different portions of video to adapt to network conditions (Sackstein, ¶¶39, 49).

186. In my opinion, a POSITA would have had a reasonable expectation of success in making this modification because Hosur already provides all of the elements needed to provide this functionality: Hosur provided multiple transcoders that would operate “concurrently,” and thus at least implicitly some way to identify

the location of portions of video across different transcoded-content stores. The only change (if any) would be expressly enabling different transcoders to transcode and store different portions of the same video.

2. Claim 1

a. 1[pre]/11[pre]: A [computer-implemented method/system] for streaming [optimized] video [to a client device], [the method] comprising:

187. To the extent the preamble is limiting, in my opinion, Hosur discloses or suggests this limitation.

188. Hosur discloses a “media-content delivery system” (cl. 11) (Hosur, ¶¶1, 14, Figs. 1, 3-4), and “method for delivering media content...to a remote client” (cl. 1) (Hosur, Abstract; *see also id.*, ¶¶6-7, 14, Fig. 1 (showing client devices 12)). Hosur explains that “various clients 12 may be capable of receiving and/or playing media associated with particular file formats, file types, codecs, bit-rates, and/or resolutions,” (*id.*, ¶18), and therefore provides a way to transcode videos “in a manner suitable for transmission to and display by the requesting client 12” (*streaming optimized video*) (*id.*, ¶¶15, 21, 25-26). Hosur also explains the method may be implemented by a computer (*computer-implemented method*). Hosur, ¶¶73-75 (transcoders including processor(s) and memory), 81-82 (other network elements may be hardware and/or “software applications being executed by [a] processor”), 83-85 (streamers including processor(s) and memory).

b. 11[a]: a processor; and

189. In my opinion, Hosur discloses or suggests this limitation.

190. Hosur discloses the system comprises *at least one processor*. Hosur, ¶¶73-75, 81-85, 89, cl. 13-23, 31-37, Figs 3-4.

c. 11[b]: a non-transitory computer-readable storage medium storing executable computer program instructions that when executed by the processor, causes the processor to:

191. In my opinion, Hosur discloses or suggests this limitation.

192. Hosur discloses the system comprises memory (*non-transitory computer-readable storage medium*) for storing *executable computer program instructions to be executed by the processor*. Hosur, ¶¶31, 34, 43, 73, 75, 80, 82-83, 85, 90, Figs. 3-4. For the system claims 11-20, as I explain in further detail below, the instructions Hosur discloses would cause the processor to perform the various functions I also address below, with or without Mattis and/or Sackstein.

d. 1[a]/11[c]: [receiving/receive] a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;

193. In my opinion, Hosur, with or without Mattis, teaches or suggests this limitation.

194. Hosur discloses that “clients 12 may include...computers.” Hosur, ¶17; *id.*, ¶18. Hosur also discloses “receiving a request identifying a requested media file”

(receiving a request [for an] original source file from a client computing device).

Hosur, ¶5; see also *id.*, ¶21 (“CMS 16 processes requests from clients 12 to download, retrieve, stream, view, or otherwise receive content.”).

195. Hosur also explains that “[t]ranscoders 20 may be capable of modifying content in source media files 30...from a first format...to a second format, such as a target format appropriate for transmission to and display by a requesting client.” Hosur, ¶26. “Content request 42 may identify a specific source media file 30” (*id.*, ¶38), as well as “indicate the appropriate format that the requested content should have,” such as a “request specifying the codec, bit-rate, and/or other parameters for the requested content” (*the request including a key specifying parameters for optimizing the original source file*) (*id.*, ¶39). “In response to receiving a content request 42, CMS 16 may...determine an appropriate format (referred to here as the ‘target format’) for the requested content to allow, facilitate, and/or optimize transmission to and playback by the requesting client” (*a request to optimize*). *Id.*, ¶39.

196. However, as I explained previously, in my opinion, a POSITA would have also supplemented or replaced Hosur’s auxiliary files with Mattis’s object/name keys. §X.B.1 (motivation). And Mattis expressly teaches including object/name keys in a request to locate a specific version of a specific file, (§X.A.2.d (Ground 1, 1[a]/11[c])), just like the parameters in Hosur’s requests. In my opinion,

when supplementing or replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, a POSITA would have understood that object/name keys in a request would be used to locate a specific version of a video file. §X.B.1 (motivation).

- e. **1[b]/11[d]: [identifying/identify], based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;**

197. In my opinion, Hosur, with or without Mattis, teaches or suggests this limitation.

198. Hosur discloses that a first portion of a specific version of a video can be identified in two ways, depending on whether the portion exists at the time of the request.

199. **The portion already exists:** Hosur explains that CMS and/or transcoder determines whether a specific version of the requested video (or portion of video) is already cached. Hosur, ¶¶40, 95. Although Hosur searches cached-content stores in one example, (*id.*), Hosur would have been understood to also search the transcoded-content stores. *Id.* That is because cached-content stores provide longer-term storage for transcoded videos after transcoding is completed. Hosur, ¶¶35-36. Hosur explains, however, that these additional stores are optional. *Id.*, ¶41 (noting the system need “not include any cached content store”). Transcoded-content stores provide storage for transcoded video during transcoding

(*id.*, ¶33), but they may also be instructed to continue storing transcoded video (*id.*, ¶59). Furthermore, Hosur explains that when a second request for transcoded content is received by the system, it directs the request to the transcoder/transcoded-content store performing the transcoding, meaning the system tracks what the transcoded-content stores contain. *Id.*, ¶¶58, 95 (noting already transcoded videos are retrieved “from an appropriate content store,” not just the cached-content store).

200. Based on these disclosures, in my opinion, Hosur would have been understood to search transcoded-content stores (*a local cache of a video optimizer*) for a first portion of transcoded video (*identifying...a first portion of an optimized version of the original source file*). In fact, other prior art (Mills) that had the exact same cached- and transcoded-content stores as Hosur and expressly searched both. Mills, Fig. 3A, ¶61 (searching both “cached content store” and “transcoded content store”).

201. Additionally, as I explain in further detail below, Hosur uses the request’s parameters and an auxiliary file with parameters to search for a specific version of video (or portions thereof) (*based on the key*). §§X.B.2.f-X.B.2.h (1[c]/11[e]-1[e]/11[g]).

202. **The portion does not already exist:** if, however, the stores do “not contain a cached media file 34...in the target format..., CMS 16 may initiate transcoding” into the target format (Hosur, ¶¶41) which was determined using the

parameters in the request (*based on the key*). *Id.*, ¶39 (“content request 42 may indicate the appropriate format”); *see also id.*, ¶¶26, 43-44. Transcoders store transcoded portions in a local transcoded-content store (*local cache of a video optimizer*). Hosur, ¶¶34, 44. To expedite video-playback, “transcoder 20 may generate a partial auxiliary file 40a...containing [location/parameter] information about a first portion of transcoded media file 32a,” which streamers may “incrementally” access to stream that portion (*identifying... a first portion*). *Id.*, ¶48; *see also id.*, ¶¶46-47, 49, 59.

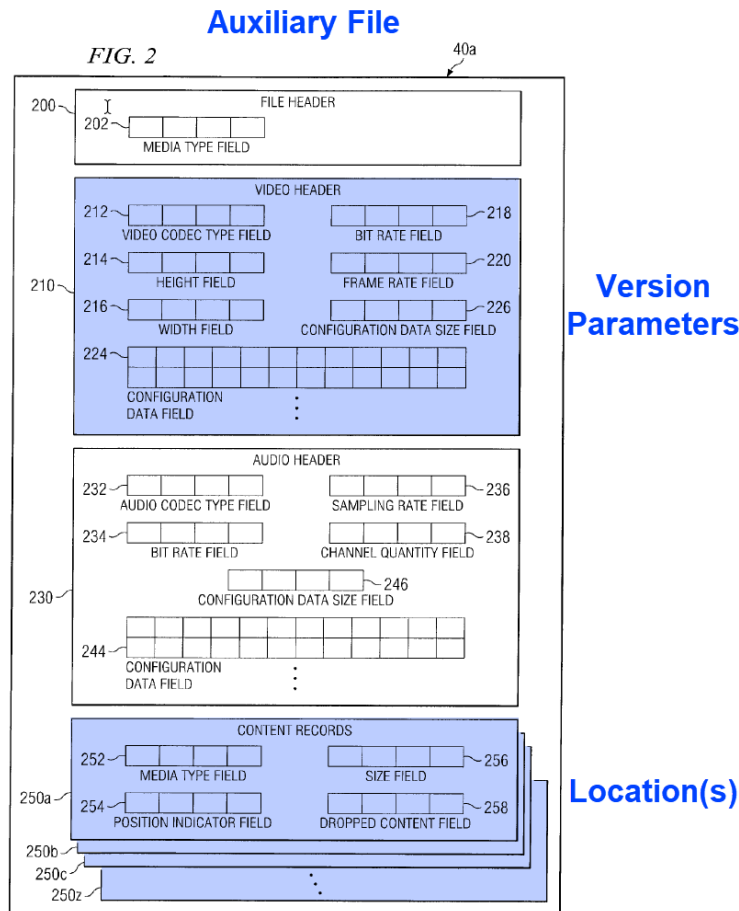
203. However, as I explained previously, in my opinion, a POSITA would have also supplemented or replaced Hosur’s auxiliary files with Mattis’s object/name keys and implemented databases like Mattis’s Tag/Directory Tables. §X.B.1 (motivation). And Mattis teaches using object/name keys in a request to locate a specific version of a multimedia object (or “fragments”) (§X.A.2.e (Ground 1, 1[b]/11[d])), just like Hosur’s requests and auxiliary files. In my opinion, when supplementing or replacing Hosur’s auxiliary files with object/name keys and databases like Mattis’s Tag/Directory Tables, a POSITA would have understood that object/name keys would be used to locate a specific version of transcoded video (or portions of video) in Hosur’s transcoded-content stores. §X.B.1 (motivation).

- f. **1[c]/11[e]: [generating/generate] a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;**

204. In my opinion, Hosur, with or without Mattis, teaches or suggests this limitation.

205. After receiving a request, Hosur queries the auxiliary file to determine whether the requested version of the video (or portions) is already cached, or has since been generated by transcoder (*generating a query*) using the request's parameters (*based on the key*). §X.B.2.e (1[b]/11[d]).

206. The “auxiliary file” identifies “a plurality of content records associated with different portions of the media content in the modified media file and indicating a location of the associated portion.” Hosur, ¶¶5-6, 46. The auxiliary file also includes, e.g., “file header information..., location data for frames or other portions..., timestamp data, framerate data, resolution data, bitrate data, and any other appropriate information” (*reference keys...specifying parameters of an optimized version of a source file*). *Id.*, ¶¶46, 62-64.



Hosur, Fig. 2

207. Hosur explains that “transcoded content store...store[s] auxiliary files 40 associated with the relevant transcoded media files.” *Id.*, ¶¶34, 35. In my opinion, by providing a collection of auxiliary files in a content store, that would broadly be considered a *database* of media files and associated auxiliary files (*a plurality of reference keys corresponding to a plurality of optimized versions of source files*). That is because a POSITA would expect the media files and associated auxiliary files to be indexed in some way for identification and retrieval.

208. Additionally, because auxiliary files “indicate a location of the associated portion of media content” (Hosur, ¶46), and there can be any number of transcoded- and cached-content stores, (Hosur, ¶31), Hosur would have been understood to teach that auxiliary files may identify other transcoded-content stores where specific versions of a video (or portions of video) are being stored (*reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers*).

209. However, even if Hosur’s collection of media and auxiliary files is not a “database,” in my opinion, a POSITA would have supplemented or replaced Hosur’s auxiliary files with Mattis’s object/name keys and implemented databases like Mattis’s Tag/Directory Tables. §X.B.1 (motivation). And Mattis teaches using the object/name key in a request to query the Tag/Directory table to locate a specific version of a multimedia object (or “fragment” portions thereof), (§X.A.2.f (Ground 1, 1[c])), just like Hosur’s request parameters and auxiliary files. In my opinion, when supplementing or replacing Hosur’s auxiliary files with object/name keys and databases like Mattis’s Tag/Directory Tables, a POSITA would have understood that a database like Mattis’s Tag/Directory Tables would be queried to search for keys matching the object/name keys in a request to locate a specific version of transcoded video (or portions thereof). §X.B.1 (motivation).

- g. 1[d]/11[f]: [receiving/receive], from the database, a response indicating whether the key matches a reference key;**

210. In my opinion, Hosur, with or without Mattis, teaches or suggests this limitation.

211. Hosur discloses that requests for a specific video and the auxiliary files associated with each stored video include parameters identifying a specific version of the video by their parameters, such as the codec used, bitrate, resolution, and more. Hosur, ¶39 (requests), 46 (auxiliary file). Either the CMS or a transcoder then “determine whether content from the requested source media file is already stored...in the target format” (*id.*, ¶¶40, 95; §X.B.2.e (1[b]/11[d])).

212. Because auxiliary files are stored in the content stores—and not some central repository—it would have been understood that the CMS or transcoder receives an indication (*receiving...a response*) from the query of an auxiliary file in the transcoded-content store (*database*) whether an auxiliary file has parameters matching those in the request (*a response indicating whether the key matches a reference key*) to determine “whether content from the requested source media file 30 is already stored...in the target format”. *Id.*, ¶¶40, 95.

213. However, as I explained previously, in my opinion, a POSITA would have supplemented or replaced Hosur’s auxiliary files with Mattis’s object/name keys and implemented databases like Mattis’s Tag/Directory Tables. §X.B.1

(motivation). And Mattis teaches receiving a response to a query of the Tag/Directory Tables indicating whether the Tables have a matching key, (§X.A.2.g (Ground 1, 1[d])), just like Hosur's request parameters and auxiliary file. In my opinion, when supplementing or replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, Hosur would receive responses indicating whether a request's key matches one in the database. §X.B.1 (motivation). This response may be received by CMS when determining whether transcoded portions already exist prior to sending a transcode instruction (Hosur, ¶40), or by transcoders/streamers when transcoding some portions of video to be combined with existing transcoded portions of video (*see, e.g., id.*, ¶49).

- h. 1[e]/11[g]: [receiving/receive] from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;**

214. In my opinion, Hosur, with or without Mattis and Sackstein, teaches or suggests this limitation.

215. As I previously explained, in my opinion, Hosur suggests that request and auxiliary-file parameters are compared during an initial search to locate existing portions of a video already stored in content stores. §X.B.2.g (1[d]/11[f]).

216. If there is no match and transcoding is required, Hosur discloses that “[s]treamer 28 receives or otherwise accesses a first record in [the] auxiliary file”

that provides “location information to access...first media portion,” and “may repeat this process for additional records” to “transmit [additional] portions.” (*receiving from the database...the address...storing a segment of the optimized version of the original source file*). Hosur, ¶¶53-54; *id.*, 70-71.

217. Hosur also discloses storing and serving different versions of video (or portions of video) from one of a plurality of transcoded-content stores. Hosur, ¶¶31 (“these content stores may each represent a plurality of devices and may be distributed across multiple locations”), 33 (noting storage of “partial file[s]”), 46 (locating portions), 48-49, 53, 55, 78. And as I explained previously, in my opinion, Hosur envisions transcoding and storing portions of a video in different transcoded-content stores. §X.B.1 (motivation). In such instances, the auxiliary file with matching parameters would provide the location of each portion in different transcoded-content stores (*in response to a reference key matching the key*). §X.B.2.g (1[d]/11[f]).

218. But as I also explained previously, even if Hosur is not read this way, Sackstein teaches transcoding and storing different portions of a given video in different transcoders. §X.B.1 (motivation) (citing by reference Sackstein, ¶¶12-13, 22, 28-35, 39-40, 47, 56-58). In my opinion, a POSITA would have been motivated to transcode and store different portions of a video in different transcoders, consistent with Sackstein, and the received address would be the address of *another*

video optimizer storing a segment of the optimized version of the original source file.

§X.B.1 (motivation).

219. Finally, as I also explained previously, a POSITA would have supplemented or replaced Hosur's auxiliary files with Mattis's object/name keys and implemented databases like Mattis's Tag/Directory Tables. §X.B.1 (motivation). And Mattis teaches that, when an object/name key in a request matches one in the Tag/Directory Tables, the location of the multimedia object (or "fragment") in one of the storage devices is provided. §X.A.2.h (Ground 1, 1[e]/11[g]). In my opinion, when supplementing or replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, the object/name key in the database would identify the address of the different transcoded-content stores (*video optimizers*) storing different fragments/portions (*storing a segment*) of the requested version of video taught by Hosur, with or without Sackstein. §X.B.1 (motivation).

- i. **1[f]/11[h]: [stitching/stitch] the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and**

220. In my opinion, Hosur, with or without Sackstein, teaches or suggests this limitation.

221. As I previously explained, Hosur suggests storing different transcoded portions of a video in different transcoded-content stores. §VI.B.1 (motivation)

(citing by reference Hosur, ¶¶15, 31, 34, 78). Hosur also discloses combining portions into a dynamic “transcoded media file” that “allow[s] streaming of content...while transcoded media file 32a...[is] still being written,” (Hosur, ¶48), and “streamers 28...synchronize portions...before streaming the relevant portions to client 12” (*id.*, ¶55). *See also id.*, ¶¶46-47, 53-56. In my opinion, therefore, Hosur contemplates “synchronizing” different portions of a video from different transcoded-content stores into a “transcoded media file” for streaming (*stitching the first portion...with the segment of the optimized version of the original source file stored on the other video optimizer*).

222. As I explained previously, however, even if Hosur is not read this way, Sackstein expressly teaches storing different portions of a video on different transcoders, searching different transcoder caches for transcoded portions of a video, and “splicing” those portions together for streaming to the client. §X.B.1 (motivation) (citing by reference Sackstein, ¶¶12-13, 22, 28-35, 39-40, 47, 56-58). When implementing Hosur to expressly store different portions of a video on different transcoders, consistent with Sackstein (§VI.B.2.h (1[e]/11[g])), a POSITA would have understood that those portions would be collected from the different transcoders and combined/spliced together for streaming to the client. §VI.B.1 (motivation).

223. Moreover, in my opinion, Hosur (with or without Sackstein) would have operated this way regardless of whether Hosur's auxiliary files were supplemented or replaced with Mattis's object/name keys and implemented databases like Mattis's Tag/Directory Tables, because that would only change *how* portions are located, and not how they are combined after the portions are located. §X.B.1 (motivation).

- j. **1[g]/11[i]: [streaming/stream] the stitched optimized version of the original source file for playback on the client computing device.**

224. In my opinion, Hosur, with or without Sackstein, teaches or suggests this limitation.

225. Hosur discloses “synchroniz[ing] portions of the modified content in transcoded media file 32a before streaming the relevant portions to client 12 ... for playback of the modified content so as to maintain synchronization during playback” (*streaming the stitched optimized version of the original source file for playback on the client device*). Hosur, ¶55. Sackstein similarly teaches stitching portions of a video and streaming those stitched portions (Sackstein, ¶40-41) for playback on the client device (*id.*, ¶33).

3. Claims 2/12

- a. 2[a]/12[a]: The [method/system] of claim [1/11], further comprising [instructions that cause the processor to]: [storing/store] a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and**

226. In my opinion, Hosur, with or without Sackstein, teaches or suggests this limitation.

227. As I explained previously, Hosur, with or without Sackstein, discloses combining different portions of transcoded video from different transcoders into a “transcoded media file.” §X.B.2.i (1[f]/11[h]). This file would be generated in either of two ways: (1) different portions would be transferred/copied to a single transcoded-content store before streaming, such that the streamer just needs to access that single file; or (2) the streamer generates the combined “transcoded media file,” in which case it would have been straightforward to save a copy of that file to a content store so that all portions would be stored together for any subsequent requests for the same transcoded video. *See, e.g.,* Ex-1033, ¶85 (“Various segments of video contents may be combined or concatenated at the time they are stored or when they are sent to the television sets for viewing.”). Either way, that would result in *storing a segment of the stitched optimized version of the original source file in the cache of the [i.e., a single] video optimizer.*

228. In fact, Hosur appears to contemplate this already. Hosur discloses that “streamer 28 may increment a pointer associated with the transcoded media file 32a based on a frame size” to “retrieve a particular video frame from a location determined based on the pointer.” Hosur, ¶54. For streamer to be able to simply “increment a pointer” to locate different portions of the video—as opposed to being instructed to look to a specific memory location—that means each portion is stored in consecutive sequences in memory; otherwise, incrementing a pointer would not point to the next portion of video, but may point to different data.

- b. 2[b]/12[b]: [transmitting/transmit], to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.**

229. In my opinion, Hosur, with or without Mattis and Sackstein, teaches or suggests this limitation.

230. Hosur discloses “transcoder 20 may generate a partial auxiliary file 40a in transcoded content store 24 containing information about a first portion of transcoded media file 32a and continue writing information to this partial auxiliary file 40a as transcoding continues.” Hosur, ¶48. Moreover, “while a transcoder 20 is transcoding...another transcoder 20 may simultaneously generate auxiliary file 40a.” Hosur, ¶47. And different transcoders may transcode different portions of a video. §X.B.2.h (1[e]/11[g]).

231. In my opinion, to enable these features, Hosur's transcoders must be able to communicate what information needs to be written into the auxiliary file, including locations, portion sizes, parameters, etc. (*transmitting...instructions to update... in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer*), including auxiliary files already stored in the transcoded-content stores (*database*).

232. However, as I also explained previously, in my opinion, a POSITA would have supplemented or replaced Hosur's auxiliary files with Mattis's object/name keys and implemented databases like Mattis's Tag/Directory Tables. §X.B.1 (motivation). And Mattis teaches updating Tag/Directory Tables when writing, moving, or deleting fragments of files. §X.A.3.b (Ground 1, 2[b]/12[b]). In my opinion, when supplementing or replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, a POSITA would have understood that the database would be updated when a new portion of video is transcoded or moved. §§X.B.3.a (2[a]/12[a]), X.B.1 (motivation).

4. Claim 3/13: The [method/system] of claim [1/11], wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.

233. In my opinion, Hosur, with or without Mattis and Sackstein, teaches or suggests this limitation.

234. Hosur, with or without Sackstein, teaches that different portions of a video may be stored in one or more transcoded-content stores. §X.B.1 (motivation). In my opinion, a POSITA would have therefore understood that querying the auxiliary files for a video would identify the different transcoders/transcoded-content stores that are storing the different portions of video so that they may be retrieved for streaming. §X.B.2.h (1[e]/11[g]).

235. However, as I also explained previously, a POSITA would have supplemented or replaced Hosur's auxiliary files with Mattis's object/name keys and implemented databases like Mattis's Tag/Directory Tables. §X.B.1 (motivation). And Mattis teaches using a matching object key in Tag/Directory Tables to locate all fragments of a requested object. §X.B.2.h (1[e]/11[g]). In my opinion, when supplementing or replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, a POSITA would have understood that a response to a matching key in a database like Mattis's Tag/Directory Tables would indicate that different portions or fragments (*segments*) of a transcoded video (*optimized version of the same original source file*) are stored in different transcoded-content stores (*a plurality of different video optimizers*). §X.B.1 (motivation).

- 5. Claims 4/14: The [method/system] of claim [3/13], further comprising [instructions that cause the processor to]: [transmitting/transmit], to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and [stitching/stitch] the different segments of the optimized version of the same original source file with the first portion of the original source file.**

236. In my opinion, Hosur, with or without Sackstein, teaches or suggests this limitation.

237. Hosur discloses that a streamer is notified that a transcoder has portion(s) of a requested video, which streamer accesses, stitches with other portions, and streams to the client. Hosur, ¶¶49, 53-55; §X.B.2.i (1[f]/11[h]). As I explained previously, Hosur, with or without Sackstein, would have been understood to store different portions of a video in different transcoded-content stores (§§X.B.2.h (1[e]/11[g]), X.B.1 (motivation)).

238. Moreover, network elements were known to operate on request-response principles. *See, e.g.*, §V (Technology Overview); Ex-1010, Fig. 1-1. In my opinion, therefore, a POSITA would have understood that Hosur's streamer would request fragments/segments from different transcoded-content stores to collect and then stream the portions stored in different transcoded-content stores. This is, for instance, exactly what Sackstein expressly describes: that "the controller 260 may

also *issue requests* for the segment to the caches of other or neighboring systems.”

Sackstein, ¶47 (emphasis added).

6. Claims 5/15: The [method/system] of claim [1/11], wherein the key comprises a source content signature, the source content signature being a hash of the original source file.

239. In my opinion, Hosur combined with Mattis teaches or suggests this limitation.

240. Hosur does not expressly disclose the use of “keys” generated through hashing, but Mattis discloses that object keys are a hash of the content of each file, and name keys are a hash of the original source file’s name each of which is a *source content signature being a hash of the original source file* (either the file itself, or its name). Mattis, 8:27-50, 8:64-9:15; *see* Ex-1001, 6:64-7:11. As I explained previously, a POSITA would have supplemented or replaced Hosur’s auxiliary files with Mattis’s object/name keys and implemented databases like Mattis’s Tag/Directory Tables. §X.B.1 (motivation).

241. If, however, the hash must be of the content of the original source file, this too would have been obvious based on Mattis as a mere design choice as to how video objects should be identified. Specifically, under Mattis’s current system, the content of the original source file would already be hashed to provide its object key. Rather than use a name key and parameters to identify a specific version of the original source file, a POSITA would have considered it an obvious alternative to

use the object key of the original source file and parameters. *See, e.g.*, Ex-1029, ¶67 (“For example, the original content ID may be generated by applying a predefined hashing or fingerprinting method to the content and using the resulting hash or fingerprint as the original content ID. The same original content ID associated with the original content received from multimedia content provider 100 may be associated with different transcoded versions of the content as may be prepared for playback by different multimedia devices. Alternatively, a different content ID may be generated by applying a predefined hashing or fingerprinting method to each transcoded version of the content and using the resulting hash or fingerprint as the content ID for its associated transcoded version. In this scenario, the content ID of each transcoded version is preferably stored by MMSC 102 in database 106 together with the original content and/or the original content ID of the original content.”). In my opinion, a POSITA would have been motivated to do so because, as Mattis recognizes, users often do not know the exact version of an object they need. Mattis, 10:10-16. But name keys also have a drawback, in that they may be too non-specific to identify a specific object when multiple *different* objects have the same name. This would be avoided by including the object key of the original source file (rather than a name key) and associated parameters in a request to identify a specific version of a specific object.

7. **Claims 6/16: The [method/system] of claim [1/11], wherein the database [comprising/comprises] data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.**

242. In my opinion, Hosur, with or without Mattis, teaches or suggests this limitation.

243. Hosur discloses that the system may store a video in “a select few frequently-requested output formats” (Hosur, ¶60), transcoding videos using different transcoders, which would thus be stored in different transcoded-content stores, and updating auxiliary files to provide their locations (*a plurality of optimization server storing an optimized version of the original source file*) (*id.*, ¶¶78; *id.*, 5-6, 46, 48, 53, 59). Hosur also discloses that auxiliary files are stored in transcoded-content stores (*database*). §X.B.2.f (1[c]/11[e]). The “auxiliary file includes a plurality of content records [(*data entries*)] associated with different portions of the media content in the modified media file and indicating a location of the associated portion” (*identifying*). Hosur, Abstract, ¶¶5, 46, 53-54, 70-71, cl. 13, 28. Finally, Hosur updates content records with new locations as new portions are transcoded (*entries being modified....*). §X.B.3.b (2[b]/12[b]).

244. However, as I also explained previously, a POSITA would have supplemented or replaced Hosur’s auxiliary files with Mattis’s object/name keys and

implemented databases like Mattis's Tag/Directory Tables. §X.B.1 (motivation). And Mattis teaches the Tag/Directory Tables (*database*) comprise entries (*data entries*) of reference keys associated with locations of fragments of different versions of a requested object, and updating Tag/Directory Table entries when fragments are written, moved, or deleted. §X.A.7 (Ground 1, cl. 6). In my opinion, when supplementing or replacing Hosur's auxiliary files with object/name keys and databases like Mattis's Tag/Directory Tables, a POSITA would have understood that the locations specified in a database like Mattis's Tag/Directory Tables would be of the transcoded-content stores storing those different versions, consistent with Hosur (*a plurality of optimization servers storing an optimized version of the original source file*). §X.B.1 (motivation).

8. **Claims 7/17: The [method/system] of claim [1/11], further comprising [instructions that cause the processor to]: [determining/determine] a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers; [transcoding/transcode], responsive to the determination, the second segment of the original source file; [storing/store] the transcoded second segment in the local cache of the video optimizer; and [stitching/stitch] the transcoded second segment with the stitched optimized version of the original source file.**

245. In my opinion, Hosur, with or without Sackstein, teaches or suggests this limitation.

246. Hosur discloses “determine[ing] whether content from the requested source media file 30 is already stored,” but if not, “initiate transcoding of a source media file 30 containing the requested content.” Hosur, ¶¶40-41. Hosur also recognizes that “a transcoded media file 32 stored in transcoded content store 24 may, at a given point in time, represent a partial file,” meaning incomplete transcoded videos may be stored. Hosur, ¶33. Sackstein similarly teaches that “a cache may be checked for transcoded segments with the same content id and similar or lower bitrates.... If such segments are found, they may be retrieved from the cache and transmitted” but “[i]f suitable segments are not found in the cache, the segments may be transcoded into their designated bitrate and the transcoded segments transmitted.” Sackstein, ¶58.

247. As I explained previously, Hosur, with or without Sackstein, would have been understood to provide multiple video transcoders capable of transcoding individual video portions upon request (§X.B.2.h (1[e]/11[g])). A POSITA would have therefore understood that, in operation, some portions of an already-transcoded video would be stored in different transcoded-content stores, but others may be missing. In this case, any missing portions would be transcoded and stitched to generate a complete transcoded video. §X.B.1 (motivation). In fact, others that implemented the same features as Hosur and Sackstein—including portioning multimedia for delivery—performed this exact feature. *See, e.g.*, Ex-1030, ¶¶42

(searching whether fragments already exist), 63 (“Unlike prepared media tracks which have all media fragments to form a complete alternative for the media, the additional alternatives may have none or only a subset of required media fragments prepared and the missing media fragments may be generated through transcoding of existing media fragments prepared for other alternatives when demanded”), Figs. 11a-b.

248. Finally, Hosur discloses that a single transcoder may transcode multiple portions of a video. Hosur, ¶¶44, 46-48. Accordingly, in some instances newly transcoded portions (e.g., *second segments*) would be stored in the same transcoded-content store as a previously transcoded first portion (*storing the transcoded second segment in the local cache of the video optimizer*).

9. **Claims 8/18: The [method/system] of claim [7/17], further comprising [instructions that cause the processor to] [transmitting/transmit] to the database, instructions [to update/for updating] the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.**

249. In my opinion, Hosur combined with Mattis, with or without Sackstein, teaches or suggests this limitation for the reasons I have provided previously. §§X.B.3.b (2[b]/12[b]), X.B.8 (cl. 7).

- 10. Claims 9/19: The [method/system] of claim [7/17], wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the second segment, that is stored by one of the plurality of other video optimizers; and transcoding the original source file from a start of the second segment to a start of the third segment.**

250. In my opinion, Hosur, with or without Sackstein, teaches or suggests this limitation.

251. Hosur, with or without Sackstein, provides multiple video transcoders capable of transcoding individual video portions upon request. §X.B.2.h (1[e]/11[g]). Additionally, both Hosur and Sackstein check whether transcoded portions/segments are already stored and transcoding missing portions/segments. §X.B.8 (cl.7).

252. For these reasons, in my opinion, a POSITA would have understood that, in operation, when a portion of video has not been transcoded (*second segment*) is between two portions that have been (*first portion and third segment*), the portion that has not yet been transcoded would be (*transcoding the original source file from a start of the second segment to a start of the third segment*) to stream a complete transcoded video to the client. §§X.B.2.h (1[e]/11[g]), X.B.8 (cl. 7); X.B.1 (motivation). In fact, others that implemented the same features as Hosur and Sackstein—including portioning multimedia for delivery—performed this exact feature. *See, e.g.*, Ex-1030, ¶¶42 (searching whether fragments already exist), 63

(“Unlike prepared media tracks which have all media fragments to form a complete alternative for the media, the additional alternatives may have none or only a subset of required media fragments prepared and the missing media fragments may be generated through transcoding of existing media fragments prepared for other alternatives when demanded”), Figs. 11a-b.

253. For instance, Hosur acknowledges that, in some instances, a portion of video may be missing or dropped (*second segment*) “between consecutive portions of transcoded media file” (e.g., *first and third segments*), which would either be part of the transcoding operation to reduce the size of the video, or due to a transcoding error. *See, e.g.*, Hosur, ¶55. Although Hosur discloses that timestamps may be used to allow the streamer/client to cover up these gaps, *id.*, it would have also been apparent that, unless a portion was purposefully dropped, the missing segment may also be re-transcoded and “synchronized” with other transcoded portions.

254. Finally, as I explained previously, because the combined system would provide multiple transcoders that may transcode different portions of the same video, a POSITA would have also understood based on Hosur and/or Sackstein that (some) already-transcoded segments would be stored in different transcoded-content stores (*third segment...that is stored by one of the plurality of other video optimizers*). §§X.B.2.h (1[e]/11[g]), X.B.8 (cl. 7); X.B.1 (motivation).

11. Claims 10/20: The [method/system] of claim [1/11], wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.

255. In my opinion, Hosur combined with Mattis, with or without Sackstein, teaches or suggests this limitation.

256. Hosur discloses deleting a transcoded video, but does not describe a detailed memory-management process for stored videos. Hosur, ¶59.

257. Mattis, however, describes a “garbage collection” process to delete unnecessary objects (or fragments/portions) from cache. Mattis, 21:54-23:52.

258. In my opinion, a POSITA would have been motivated to modify Hosur to implement a “garbage collection” process like Mattis’s to manage available memory in Hosur’s transcoders (*video optimizers*). In doing so, a POSITA would have expected that a first portions/fragments of a video would be more-frequently accessed than later fragments. This is because it was well-known that end users would often start, but not complete playback of an entire video. Ex-1031, ¶34 (“many people will often start watching a video at its beginning, but will stop watching after a relatively brief period of time.”). Because Mattis implements a garbage-collection process that is fragment-specific and based on frequency of access, then, in many instances later fragments would be deleted (for being less frequently accessed) but the first portion would not. In fact, Sackstein describes a similar deletion process to ensure sufficient cache memory remains available for

newly transcoded portions of video, deleting old portions based on frequency of use and/or popularity. Sackstein, ¶¶45, 59.

259. Although Mattis notes that if one fragment of a multi-fragment object is marked for deletion, all fragments are similarly marked for deletion (Mattis, 23:24-31), in my opinion, a POSITA would have understood that Mattis did so because Mattis provided no way to re-create a deleted fragment, meaning deleting one fragment would likely corrupt the object. However, when providing transcoders capable of recreating deleted portions/fragments (like Hosur does), a POSITA would have appreciated that there would be no need to delete all fragments if some subset are sufficiently active/popular and others are not, because inactive portions would simply be re-transcoded. This is again consistent with Sackstein, which envisioned deleting some, but not all, segments of a video based on popularity of individual segments. Sackstein, ¶¶45, 59.

C. Ground 3A: Mills in view of Mattis, with or without Sackstein

1. Motivation to Combine

260. In my opinion, a POSITA would have been motivated to combine Mills, Mattis, and Sackstein because each of these three references are directed to the same field of content-delivery networks that store and deliver different versions of multimedia content to clients, just like the '442 patent. *See* §§IX.1 (Mattis), IX.3 (Mills), IX.4 (Sackstein); §VI.A ('442 Overview). Moreover, all three references

implicate similar principles, and already share many similar features, but focus on slightly different aspects of these delivery systems.

261. Mills, for instance, is focused on generating new versions of multimedia (as well as ways to locate different versions of multimedia) to expedite playback time; Mattis is focused on how different versions of a multimedia are stored and located in a network; and Sackstein is focused on generating new versions of multimedia to account for bandwidth constraints in a network. Because each had overlapping processes with slightly different focuses, their teachings would have been recognized as largely complementary of one another with minimal modification.

262. A POSITA would have been specifically motivated to modify Mills in a few ways.

263. **First**, like Hosur, Mills explains how requests may include parameters about the client, which are used by Mills's system to identify a specific version of video to stream in response to a client's request. Mills, ¶33 ("CMS 16 may receive an HTTP request specifying the codec, bit-rate, and/or other parameters for the requested media file 30. As a result, CMS 16 may be able to determine, based on information contained in the HTTP request, the output format and characteristics appropriate for the requested media file 30"). Mills does not, however, explain how the parameters are used to determine whether the video already exists into the

appropriate format matching the parameters, or whether it needs to be transcoded to respond to the client's request. Mills left these implementation details to skilled artisans.

264. In my opinion, Mattis would have been recognized as providing one obvious option for implementing Mills's parameter-based searching. Specifically, Mattis describes a system that—like Mills—also uses parameters in a request as part of a “name key.” *See* §§IX.1 (Mattis Overview), X.A.2.e-X.A.2.f (Ground 1, 1[b]/11[d]-1[c]/11[e]). But, unlike Mills, Mattis provides a detailed description as to how parameters are used to determine whether a file already exists in the requested format: by comparing name keys and parameters to reference name keys and parameters in Tag/Directory Tables. *Id.* In addition to parameter-based searching, Mattis also provides an additional “object key” to detect duplicate objects and request a specific version of a specific file directly. *Id.*

265. In my opinion, a POSITA would have been motivated to combine Mills and Mattis by implementing Mattis's object/name keys to locate specific versions of videos in a way that is already consistent with Mills. As part of that combination, a POSITA would have been motivated to include object and name keys in requests, as well as databases like Mattis's Tag/Directory Tables to both determine whether the requested version of video already exists in the system and avoid storing duplicative

videos, thus reducing memory and processing requirements. Mattis, 8:51-9:15, 10:10-22.

266. Regardless, in my opinion, implementing Mattis's keys and database tables in Mills's system would have merely amounted to a simple substitution of one known element (Mills's summary use of "parameters") for another (Mattis's detailed use of object/name keys and Tag/Directory Tables) to achieve a predictable result (locating portions of a specific version of a specific video). It also would have amounted to replacing one known technique (Mills's use of parameters to identify specific versions of video) with another suitable option (Mattis's use of name/object keys and Tag/Directory Tables) to solve a known problem in the art (locating a specific version of a requested video).

267. In my opinion, a POSITA would have had a reasonable expectation of success in making this modification because Mills and Mattis are complementary of one another, and the changes would be made without altering the underlying functionality of each. For instance, Mills already suggests using what Mattis calls a "name key" to identify the requested video, as well as the parameters associated with a specific version of the requested video. Mattis simply provides additional details to carry out searches using those parameters, as well as an alternative type of (object) key to request specific versions directly. In fact, the only modifications required would be to incorporate Mattis's keys, databases, and queries, which would have

been well within the skills of a POSITA. The remainder of Mills's system would still operate exactly as Mills describes (e.g., receiving requests, locating responsive videos, transcoding when necessary, and streaming the responsive video), and Mattis's keys/databases would likewise operate exactly as Mattis describes (including keys in requests, and using those keys to search a database to locate responsive video or determine a responsive version of video is not present).

268. **Second**, when explaining the operation of the invention, Mills describes transcoding and storing portions of a given video in a single transcoder. Mills, ¶¶36-41. In my opinion, however, Mills was not limited to transcoding a video using a single transcoder. Mills explains, for instance, that the network may be provided with multiple transcoders/transcoding modules that may operate “concurrently so that multiple sets of content can be modified simultaneously” (Mills, ¶53), and enable “realtime transcoding of media files 30 to prevent the streaming transcoder 24 from exhausting the available transcoded content” (*id.*, ¶24). Mills also envisions that “[t]ranscoded content store 24 stores media files 30 *or portions of medial files 30 after the content has been transcoded by transcoders.*” *Id.*, ¶23 (emphasis added). And Mills also discloses streaming from a partial “accessed media file” while portions of transcoded video are still being written to the file. *Id.*, ¶43.

269. In my opinion, Mills envisioned a system capable of having different transcoders transcode different portions of the same video. For instance, transcoding is processor intensive. Depending on the extent of the transcoding, it can take longer to convert the video than to play back the entire video. Mills's desire to provide "realtime transcoding"—where the client cannot playback portions faster than they are being transcoded—would have therefore counseled using multiple transcoders to simultaneously transcode different portions of the same video to ensure transcoding was not a bottleneck for playback. *See* Ex-1025, Abstract, 1-5.

270. Separately, it was common for clients to request video but then stop playback before completing the video. In these cases, there are only a few choices: (1) use limited processing power to complete transcoding even though the client no longer demands it for playback; (2) delete the partially transcoded video, and then redo the same transcoding operations when another client requests the same video; or (3) store the already transcoded portions, and when the same client or a different client requests the same transcoded version of video, transcode only those portions that were not previously transcoded. Given Mills's focus on transcoding and storing portions of transcoded video, a POSITA would have appreciated that Mills at least contemplated, if not expressly directed, to the third option.

271. For these reasons, in my opinion, Mills would have provided a system with multiple transcoders that would both simultaneously transcode different

portions of the same video for faster-than-real-time transcoding, as well as store partially transcoded videos in response to one request, and resume transcoding using the same or different transcoders upon subsequent requests.

272. However, even if Mills would not be read this way, in my opinion, a POSITA would have been expressly motivated to modify Mills to provide these features based on Sackstein. Sackstein describes a process of segmenting a video for transcoding, have a distributed set of transcoders simultaneously transcode the video and store these portions in their caches, and then locate and splice the different portions together. Sackstein, ¶¶12-13, 27-28, 30-33, 39-40, 47, 56-58. A POSITA would have recognized that this provides two benefits: (1) it provides faster transcoding of the complete video (Ex-1025, Abstract, 1-5); and (2) it allows different transcoders to apply different parameters to different portions of video to adapt to network conditions (Sackstein, ¶¶39, 49).

273. In my opinion, a POSITA would have had a reasonable expectation of success in making this modification because Mills, and especially when combined with Mattis, already provided all of the elements needed to provide this functionality: Mills provided multiple transcoders that would operate “concurrently” and would store separate “portions” of video, and Mattis provided storing and locating fragments/portions of objects in different memory stores. The only apparent

change (if any) would be expressly enabling different transcoders to transcode and store different portions of the same video.

2. Claim 1/11

a. 1[pre]/11[pre]: A [computer-implemented method/system] for streaming [optimized] video [to a client device], [the method] comprising:

274. To the extent the preamble is limiting, in my opinion, Mills discloses or suggests it.

275. Mills discloses a “system [(cl. 11)] and method [(cl. 1)] for managing and modifying media content,” (Mills, ¶1), and “delivering content from media files...to clients 12...of different reception and display requirements,” including video (*a method for streaming [optimized] video [to a client device]*) (Mills, ¶12; *id.*, ¶¶14-15, 21, 26, 43-45 (streaming), Figs. 3A-B). Mills also explains the method may be implemented by a computer. Mills, ¶¶49-50, 57 (transcoders including processor(s) and memory), 56 (other network elements may be hardware and/or “software applications being executed by [a] processor”).

b. 11[a]: a processor; and

276. In my opinion, Mills discloses or suggests this limitation.

277. Mills discloses the system comprises *a processor*. Mills, ¶¶49-50, 56-57.

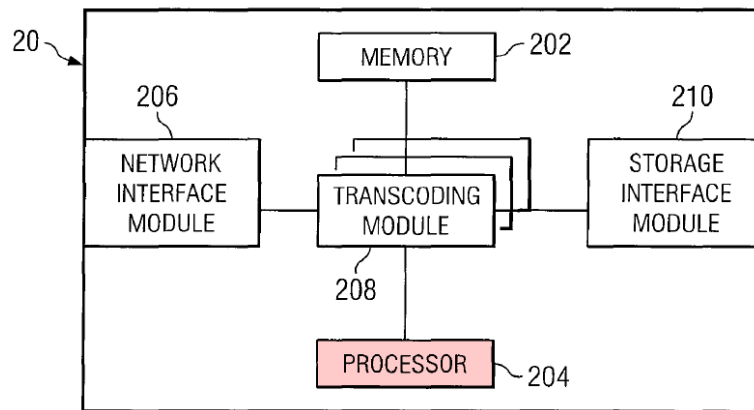


FIG. 2

Mills, Fig. 2

- c. **11[b]: a non-transitory computer-readable storage medium storing executable computer program instructions that when executed by the processor, causes the processor to:**

278. In my opinion, Mills discloses or suggests this limitation.

279. Mills discloses the system comprises memory (*non-transitory computer-readable storage medium*) for storing *executable computer program instructions to be executed by the processor*. Mills, ¶51; *id.*, ¶¶22-23, 25, 40, 49, 55.

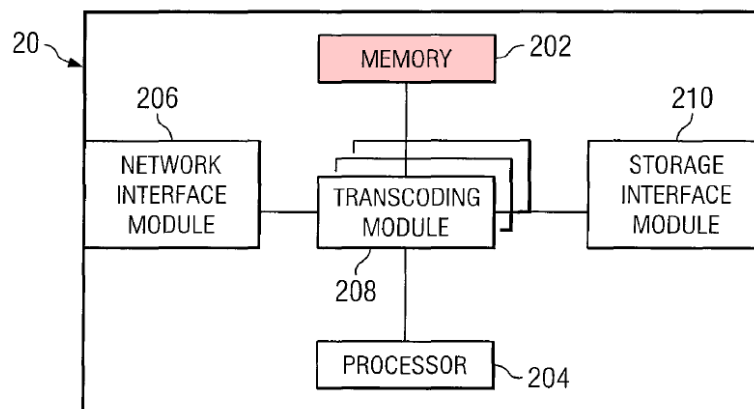


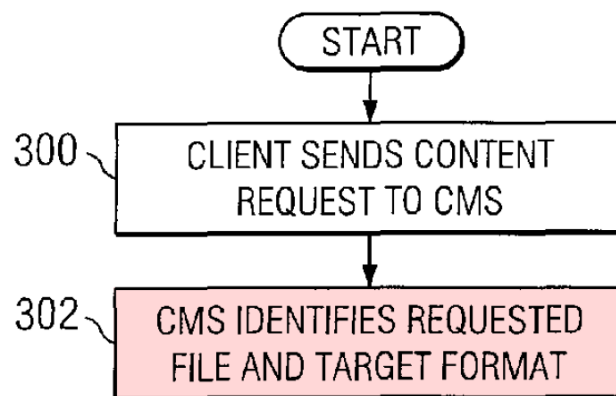
FIG. 2

For the system claims 11-20, as I explain in further detail below, the instructions Mills discloses would cause the processor to perform the various functions I also address below, whether with or without Mattis and/or Sackstein.

d. 1[a]/11[c]: [receiving/receive] a request to optimize an original source file from a client computing device, the request including a key specifying parameters to optimize the original source file;

280. In my opinion, Mills, with or without Mattis, teaches or suggests this limitation.

281. Mills discloses that “clients 12 may include...computers.” Mills, ¶14; *id.*, ¶15. Mills also discloses “receiving a request identifying a media file from a first client...in a first format,” and “modifying the media file...to a second format associated with the first client” (*receiving a request to optimize an original source file from a client computing device*). Mills, ¶5; *id.*, ¶¶6, 13, 59 (“Operation...begins at step 300 with a client 12 sending a content request 32 for a media file 30”).



Mills, Fig. 3A

282. Mills also explains that the request may “specify[] the codec, bit-rate, and/or other parameters for the requested media file 30” to determine “the output format and characteristics appropriate for the requested media file 30” (*the request including a key specifying parameters for optimizing the original source file*). Mills, ¶33.

283. However, as I explained previously, a POSITA would have implemented Mattis’s object/name keys in Mills’s system. §X.C.1 (motivation). And Mattis expressly teaches including object/name keys in a request to locate a specific version of a specific file, (§X.A.2.d (Ground 1, 1[a]/11[c])), just like Mills’s parameter-based search. When implementing object/name keys and a database like Mattis’s Tag/Directory Tables in Mills, a POSITA would have understood that the object/name keys in a request would be used to locate a specific version of a video file. §X.C.1 (motivation).

- e. **1[b]/11[d]: [identifying/identify], based on the key, a first portion of an optimized version of the original source file in a local cache of a video optimizer;**

284. In my opinion, Mills, with or without Mattis, teaches or suggests this limitation.

285. Mills discloses that a first portion of a specific version of a video can be identified in two ways, depending on whether the portion exists at the time of the request.

286. **The portion already exists:** Mills explains that, first, “load balancing server 18 may...determine at step 308 whether the requested media file 30 is available in transcoded content store 24 in the target format,” (Mills, ¶¶60-61, Fig. 3A (308)), and that videos may be transcoded/stored as portions within the transcoded-content store (*id.*, ¶63) (*identifying...a first portion of an optimized version of the original source file in a local cache of a video optimizer*). Mills, ¶¶60-61, 67, Fig. 3A (308).

287. Additionally, as I explain in further detail below, Mills uses the request’s parameters to search for a specific version of video (or portions thereof) (*based on the key*). §§X.C.2.f-X.C.2.h (1[c]/11[e]-1[e]/11[g]).

288. **The portion does not already exist:** if, however, the requested version of video is not already cached, “load balancing server 18 may then select a transcoder 20 to modify requested media file 30 according to the target format.” Mills, ¶62. “Before or after initiating this modification process, transcoder 20 may send a transcode response 36 to CMS 16...indicat[ing] that...a portion of the modified content is available” in the transcoded-content store (*identifying...a first portion of an optimized version of the original-source file*). *Id.*, ¶64. Moreover, Mills explains that “transcoded content store 24 may represent storage local or internal to transcoders 20” (*a local cache of a video optimizer*) (*id.*, ¶23), and the target format

is determined based on the parameters provided in a request (*based on the key*) (*id.*, ¶33). *See also id.*, ¶¶30-31, 40-41, Fig. 3A (318-326).

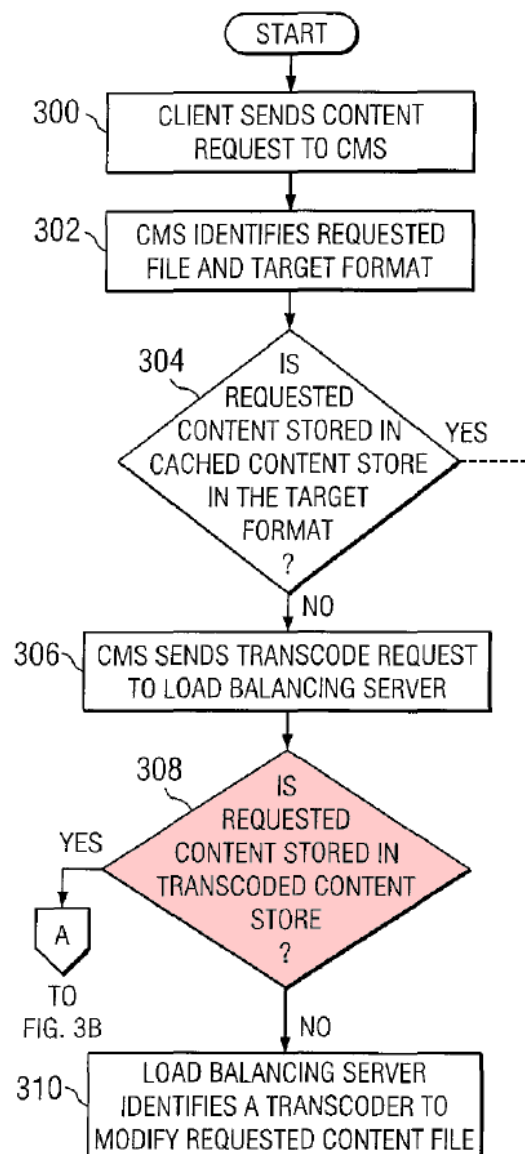
289. However, as I explained previously, in my opinion, a POSITA would have implemented Mattis's object/name keys in Mills's system. §X.C.1 (motivation). And Mattis teaches using object/name keys in a request to locate a specific version of a multimedia object (or "fragments") (§X.A.2.e (Ground 1, 1[b]/11[d]), just like Mills's parameter-based search. When implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, a POSITA would have understood that the object/name keys would be used to locate a specific version of transcoded video (or portions of video). §X.C.1 (motivation).

- f. **1[c]/11[e]: [generating/generate] a query, based on the key, to a database maintaining a plurality of reference keys corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file;**

290. In my opinion, Mills combined with Mattis teaches or suggests this limitation.

291. Mills discloses that, after receiving a request, the system queries the transcoded-content stores to determine whether the requested video (or portions of video) is already cached, or has since been generated by a transcoder (*generating a query*) using the request's parameters (*based on the key*). §X.C.2.e (1[b]/11[d]);

Mills, ¶33. Mills also explains that different versions of video (or portions of video) may be stored in multiple locations, including different transcoded-content stores (*plurality of optimized versions of source files stored on a plurality of other video optimizers*). Mills, ¶¶23, 53-54.



Mills, Fig. 3A

292. Mills does not specify how the request's parameters are used to search the stores. Mills, ¶33. Mattis, however, provides one way to carry out Mills's searches by using a request's object/name key to query the Tag/Directory Tables (*a database maintaining a plurality of reference keys to locate a specific version of a multimedia object (or "fragments") (corresponding to a plurality of optimized versions of source files stored on a plurality of other video optimizers, each reference key specifying parameters of an optimized version of a source file)*) (§X.A.2.f (Ground 1, 1[c]/11[e])), just like Mills's nondescript parameter-based search. In my opinion, when implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, a POSITA would have understood that the database would be queried to search for keys matching the request's object/name key to locate a specific version of video (or portions of video). §X.C.1 (motivation).

- g. 1[d]/11[f]: [receiving/receive], from the database, a response indicating whether the key matches a reference key;**

293. In my opinion, Mills combined with Mattis teaches or suggests this limitation.

294. Mills discloses that CMS searches the cached-content store to determine whether a specific version of a video already exists (Mills, ¶60), and the load-balancing server searches the transcoded-content store to determine the same (*id.*, ¶61). As I mentioned previously, Mills does not specify how the request's

parameters are used to search the stores. Mills, ¶33; §X.C.2.f (1[c]/11[e]). Mattis, however, provides one way to carry out those searches (§X.C.1 (motivation)), and teaches receiving a response from the Tag/Directory Tables indicating whether the request's key matches a reference key in the Tables to locate a specific version of a multimedia object (or "fragments") (*receiving, from the database, a response indicating whether the key matches a reference key*). §X.A.2.g (Ground 1, 1[d]/11[f]).

295. In my opinion, when implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, a POSITA would have understood that Mills's CMS or load-balancing server would receive responses from the database indicating whether request's key matches a reference key in the Tables. §X.C.1 (motivation). Specifically, despite Mills breaking up the search across two components (CMS and LBS), a POSITA would have understood the search would be simplified by using only one component to search both classes of stores (cached- and transcoded-content stores), and/or eliminating one class of stores—as Hosur proposes for the cached-content stores (Hosur, ¶41)—to simplify the storage and search scheme.

- h. 1[e]/11[g]: [receiving/receive] from the database, in response to a reference key matching the key, the address of another video optimizer storing a segment of the requested optimized version of the original source file;**

296. In my opinion, Mills combined with Mattis, with or without Sackstein, teaches or suggests this limitation.

297. Mills discloses that the request's parameters are used to locate a specific version of video. Mills, ¶33. Mills also explains that transcoded-content stores may store "portions of media files 30." *Id.*, ¶23. And, for the reasons I previously provided, Mills would have been understood to envision transcoding and storing different portions of a specific version of video in different transcoders and their respective transcoded-content stores. §X.C.1 (motivation).

298. As I explained previously, a POSITA would have implemented Mattis's object/name keys, as well as databases like Mattis's Tag/Directory Tables into Mills's system to carry out Mills's searches. §X.C.1 (motivation). And Mattis teaches that, when an object/name key in a request matches one in the Tag/Directory Tables, the location (*address*) of one of a plurality of storage devices storing the specific version of a multimedia object (or "fragment" portions thereof) is provided. §X.A.2.h (Ground 1, 1[e]/11[g]). In my opinion, when implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, a POSITA would have understood that the database would issue responses with the location (*address*)

of a specific version of video (or portions) in different transcoded-content stores.
§X.C.1 (motivation).

299. Moreover, even if Mills is not read as transcoding and storing different portions of a specific version of video in different transcoders and their respective transcoded-content stores, Sackstein expressly teaches transcoding and storing different portions of a given video in different transcoders as part of concurrent transcoding. §X.C.1 (motivation) (citing by reference Sackstein, ¶¶12-13, 22, 28-35, 39-40, 47, 56-58). In my opinion, a POSITA would have been motivated to transcode and store different portions of a video in different transcoders, consistent with Sackstein, to enable faster transcoding as well as dynamic adaptation of individual portions of video. §X.C.1 (motivation). Moreover, upon making this additional combination, a POSITA would have understood that the *address* received by comparing keys in the request to keys in Mills-Mattis's database would be the address of *another video optimizer storing a segment of the requested optimized version of the original source file*. §X.C.1 (motivation).

- i. **1[f]/11[h]: [stitching/stitch] the first portion of the optimized version of the original source file with the segment of the optimized version of the original source file stored on the other video optimizer; and**

300. In my opinion, Mills, with or without Sackstein, teaches or suggests this limitation.

301. As I previously explained, Mills suggests storing different transcoded portions of a video in different transcoded-content stores. §X.C.1 (motivation) (citing by reference Mills, ¶¶23-24, 38, 53). Mills also discloses combining portions into a dynamic “created media file 30b” that allows “reading transcoded content out of the created media file 30b at the same time transcoder 24 is writing to media file 30b.” Mills, ¶24. In my opinion, therefore, a POSITA would have understood Mills as contemplating that different portions of the same transcoded video would be combined from different transcoded-content stores that had each transcoded a relevant portion to generate the “created media file” for streaming (*stitching the first portion...with the segment of the optimized version of the original source file stored on the other video optimizer*).

302. As I explained previously, however, even if Mills is not read this way, Sackstein expressly teaches storing different portions of a video on different transcoders, searching different transcoder caches for transcoded portions of a video, and “splicing” those portions together for streaming to the client. §X.C.1 (motivation) (citing by reference Sackstein, ¶¶12-13, 22, 28-35, 39-40, 47, 56-58). When implementing Mills to expressly store different portions of a video on different transcoders, consistent with Sackstein (§VI.B.2.h (1[e]/11[g])), a POSITA would have understood that those portions would be collected from the different

transcoders and combined/spliced together for streaming to the client. §X.C.1 (motivation).

303. Moreover, in my opinion, Mills (with or without Sackstein) would have operated this way regardless of whether Mills implemented Mattis’s object/name keys and databases like Mattis’s Tag/Directory Tables, because that would only change *how* portions are located, and not how they are combined after the portions are located. §X.C.1 (motivation).

j. 1[g]/11[i]: [streaming/stream] the stitched optimized version of the original source file for playback on the client computing device.

304. In my opinion, Mills, with or without Sackstein, teaches or suggests this limitation.

305. Mills discloses streaming the contents of the transcoded (“created”) media file to enable “playback by clients” (*streaming the stitched optimized version of the original source file for playback on the client computing device*). Mills, ¶29; *id.*, ¶¶28 (playback), 33 (transcoding parameters for target format), 43 (“As an alternative to or in addition to transcoder 24 transmitting the requested media file 30 to a particular client 12, particular embodiments of system 10 may include streaming elements”), 65, 69 (transmission/streaming). Sackstein similarly teaches stitching portions of a video and streaming those stitched portions (Sackstein, ¶40-41) for playback on the client device (*id.*, ¶33).

3. Claims 2/12

- a. 2[a]/12[a]: The [method/system] of claim [1/11], further comprising [instructions that cause the processor to]: [storing/store] a segment of the stitched optimized version of the original source file in a cache of the video optimizer; and**

306. In my opinion, Mills, with or without Sackstein, teaches or suggests this limitation.

307. As I explained previously, Mills, with or without Sackstein, discloses combining different portions of transcoded video from different transcoders into a “transcoded media file.” §X.C.2.i (1[f]/11[h]). This file would be generated in either of two ways: (1) different portions would be transferred/copied to a single transcoded-content store before streaming, such that the streamer just needs to access that single file; or (2) the transcoder/streamer generates the combined “created media file,” in which case it would have been straightforward to save a copy of that file to a content store so that all portions would be stored together for any subsequent requests for the same transcoded video. *See, e.g.,* Ex-1033, ¶85 (“Various segments of video contents may be combined or concatenated at the time they are stored or when they are sent to the television sets for viewing.”). Either way, that would result in *storing a segment of the stitched optimized version of the original source file in the cache of the [i.e., a single] video optimizer.*

308. In fact, Mills discloses that, after transcoding/delivery to the client is complete, the entire modified media file may be stored to fulfill subsequent requests. Mills, ¶66. Although Mills stores it in the cached-content store (*id.*), a POSITA would have recognized that it would have been straightforward to store it in a transcoded-content store instead.

309. Specifically, a POSITA would have appreciated that (like Hosur), Mills intended for the cached-content store to provide longer-term storage of content than the transcoded-content store. Mills, ¶¶21 (“cached content store 26 stores media files 30 in a commonly-requested format to minimize the frequency with which transcoding is performed”), 36 (same), 47 (“transcoder 20 may send a message to the CMS 16 in response to which CMS 16 may direct transcoder 20 to continue storing the modified content in transcoded content store 24, send the modified content to cached content store 26, and/or delete modified content from the transcoded content store 24”). However, a POSITA would have appreciated that both content stores are just the same types of memory in different structures/locations. Mills, ¶25 (“original content store 22, transcoded content store 24, and cached content store 26 may each represent or include any appropriate type of memory devices”).

310. Rather than have two separate stores that would need to be searched (separately), a POSITA would have recognized that Mills would be readily

simplified by providing a single memory store within transcoders (in addition to the “original content store” typically owned by the content provider). For instance, Hosur taught the same three stores (original-, cached-, and transcoded-content stores) as Mills, but expressly noted that cached-content stores were optional. Hosur, ¶41 (noting system may “not include any cached content store 26”). This would have been a recognized design choice as to how the memory of the network was structured: transcoders and memory could be separated, with different memory stores serving different purposes, or integrating it all together into a single structure.

b. 2[b]/12[b]: [transmitting/transmit], to the database, instructions to update the database in response to storing the segment of the stitched optimized version of the original source file stored in the video optimizer.

311. In my opinion, Mills combined with Mattis, with or without Sackstein, teaches or suggests this limitation.

312. Mills discloses that, after transcoding a portion of video, “transcoder 20 may send transcode response 36 to load balancing server 18 and/or CMS 16 via network 14 indicating that a portion of the requested media file 30 has been transcoded” and is available for streaming. Mills, ¶41. Mills does not, however, specify how newly generated versions of a video (or portions) are located.

313. Mattis, however, teaches updating Tag/Directory Tables when writing, moving, or deleting fragments of files. §X.A.3.b (Ground 1, 2[b]/12[b]). In my

opinion, when implementing object/name keys and a database like Mattis's Tag/Directory Tables in Mills, a POSITA would have understood that the database would be updated when collecting all portions together in a "created media file," as well as storing the complete "created media file" after delivery. §§X.C.3.a (2[a]/12[a]), X.C.1 (motivation).

4. Claim 3/13: The [method/system] of claim [1/11], wherein the received response indicates that different segments of the optimized version of the same original source file are stored on a plurality of different video optimizers.

314. In my opinion, Mills combined with Mattis, with or without Sackstein, teaches or suggests this limitation.

315. Mills, with or without Sackstein, teaches that different portions of a video may be stored in one or more transcoded-content stores. §X.C.1 (motivation). And Mattis also teaches using a matching object key in Tag/Directory Tables to locate all fragments of a requested object. §VI.A.5.h (Ground 1, 1[e]/11[g]).

316. As I explained previously, a POSITA would have implemented object/name keys and databases like Mattis's Tag/Directory Tables in Mills's system. §X.C.1 (motivation). In my opinion, when implementing object/name keys and databases like Mattis's Tag/Directory Tables in Mills's system, a POSITA would have understood that a response to a matching key in a database like Mattis's Tag/Directory Tables would indicate that different portions or fragments (*segments*)

of a transcoded video (*optimized version of the same original source file*) are stored in different transcoded-content stores (*a plurality of different video optimizers*).

§X.C.1 (motivation).

5. **Claims 4/14: The [method/system] of claim [3/13], further comprising [instructions that cause the processor to]: [transmitting/transmit], to the plurality of different video optimizers, a request for the segment of the optimized version of the original source file stored on each video optimizer; and [stitching/stitch] the different segments of the optimized version of the same original source file with the first portion of the original source file.**

317. In my opinion, Mills combined with Mattis, with or without Sackstein, teaches or suggests this limitation.

318. Mills discloses that a load-balancing server or CMS is notified that a transcoder has portion(s) of a requested video, which either a transcoder handling transmission, or separate streamer, accesses and streams to the client. Mills, ¶¶24, 41-43, 45; §X.C.2.i (1[f]/11[h]). As I explained previously, Mills, with or without Sackstein, would have been understood to store different portions of a video in different transcoded-content stores. §§X.C.2.h (1[e]/11[g]), X.C.1 (motivation).

319. Moreover, network elements were known to operate on request-response principles. *See, e.g.*, §V (Technology Overview); Ex-1010, Fig. 1-1. As such, Mills's transcoder/streamer would request portions from different transcoded-content stores to collect and then stream the portions stored in different transcoded-

content stores. This is, for instance, exactly what Sackstein expressly describes: that “the controller 260 may also *issue requests* for the segment to the caches of other or neighboring systems.” Sackstein, ¶47 (emphasis added).

6. Claims 5/15: The [method/system] of claim [1/11], wherein the key comprises a source content signature, the source content signature being a hash of the original source file.

320. In my opinion, Mills combined with Mattis teaches or suggests this limitation.

321. Mills does not expressly disclose the use of “keys” generated through hashing, but Mattis discloses that object keys are a hash of the content of each file, and name keys are a hash of the original source file’s name each of which is a *source content signature being a hash of the original source file* (either the file itself, or its name). Mattis, 8:27-50, 8:64-9:15; *see* Ex-1001, 6:64-7:11; §X.C.1 (motivation).

322. If, however, the hash must be of the content of the original source file, this too would have been obvious based on Mattis as a mere design choice as to how video objects should be identified. Specifically, under Mattis’s current system, the content of the original source file would already be hashed to provide its object key. Rather than use a name key and parameters to identify a specific version of the original source file, a POSITA would have considered it an obvious alternative to use the object key of the original source file and parameters. *See, e.g.*, Ex-1029 ¶67 (“For example, the original content ID may be generated by applying a predefined

hashing or fingerprinting method to the content and using the resulting hash or fingerprint as the original content ID. The same original content ID associated with the original content received from multimedia content provider 100 may be associated with different transcoded versions of the content as may be prepared for playback by different multimedia devices. Alternatively, a different content ID may be generated by applying a predefined hashing or fingerprinting method to each transcoded version of the content and using the resulting hash or fingerprint as the content ID for its associated transcoded version. In this scenario, the content ID of each transcoded version is preferably stored by MMSC 102 in database 106 together with the original content and/or the original content ID of the original content.”). In my opinion, a POSITA would have been motivated to do so because, as Mattis recognizes, users often do not know the exact version of an object they need. Mattis, 10:10-16. But name keys also have a drawback, in that they may be too non-specific to identify a specific object when multiple *different* objects have the same name. This would be avoided by including the object key of the original source file (rather than a name key) and associated parameters in a request to identify a specific version of a specific object.

7. **Claims 6/16: The [method/system] of claim [1/11], wherein the database [comprising/comprises] data entries identifying a plurality of video optimizers storing an optimized version of the original source file, the entries being modified through instructions transmitted to the database to update a location where the optimized version of the original source file is stored.**

323. In my opinion, Mills combined with Mattis teaches or suggests this limitation.

324. Mills discloses that the system may store a video in “a select few frequently requested output formats” (Mills, ¶48), and transcoding videos using different transcoders, which would thus be stored in different transcoded-content stores (*a plurality of optimization server storing an optimized version of the original source file*) (*id.*, ¶53; *id.*, ¶¶5-6, 54). Mills explains that a request’s parameters are used to locate a specific version of video, (Mills, ¶33), but does not specify how.

325. However, as I explained previously, in my opinion, a POSITA would have implemented Mattis’s object/name keys, as well as databases like Mattis’s Tag/Directory Tables into Mills’s system to carry out Mills’s searches. §X.C.1 (motivation). And Mattis teaches the Tag/Directory Tables (*database*) comprise entries (*data entries*) of reference keys associated with locations of fragments of different versions of a requested object, and updating Tag/Directory Table entries when fragments are written, moved, or deleted. §X.A.7 (Ground 1, cl. 6). In my opinion, when implementing object/name keys and databases like Mattis’s

Tag/Directory Tables in Mills's, a POSITA would have understood that the locations specified in a database like Mattis's Tag/Directory Tables would be of the transcoded-content stores storing those different versions, consistent with Mills (*a plurality of optimization servers storing an optimized version of the original source file*). §X.C.1 (motivation).

8. **Claims 7/17: The [method/system] of claim [1/11], further comprising [instructions that cause the processor to]: [determining/determine] a second segment of the optimized version of the original source file is not stored by the plurality of other video optimizers; [transcoding/transcode], responsive to the determination, the second segment of the original source file; [storing/store] the transcoded second segment in the local cache of the video optimizer; and [stitching/stitch] the transcoded second segment with the stitched optimized version of the original source file.**

326. In my opinion, Mills, with or without Sackstein, teaches or suggests this limitation.

327. Mills discloses determining “whether the requested media file 30 is already stored” but if not, “initiate transcoding of a copy of the requested media file.” Mills, ¶¶36-37. Mills also recognizes that “[t]ranscoded content store 24 stores media files 30 or portions of media files 30.” Mills, ¶23. Sackstein similarly teaches that “a cache may be checked for transcoded segments with the same content id and similar or lower bitrates.... If such segments are found, they may be retrieved from the cache and transmitted” but “[i]f suitable segments are not found in the cache, the

segments may be transcoded into their designated bitrate and the transcoded segments transmitted.” Sackstein, ¶58.

328. As I explained previously, Mills, with or without Sackstein, would have been understood to provide multiple video transcoders capable of transcoding individual video portions upon request (§X.C.2.h (1[e]/11[g])). A POSITA would have therefore understood that, in operation, some portions of an already-transcoded video would be stored in different transcoded-content stores, but others may be missing. In this case, any missing portions would be transcoded and stitched to generate a complete transcoded video. §X.C.1 (motivation). In fact, others that implemented the same features as Mills and Sackstein—including portioning multimedia for delivery—performed this exact feature. *See, e.g.*, Ex-1030, ¶¶42 (searching whether fragments already exist), 63 (“Unlike prepared media tracks which have all media fragments to form a complete alternative for the media, the additional alternatives may have none or only a subset of required media fragments prepared and the missing media fragments may be generated through transcoding of existing media fragments prepared for other alternatives when demanded”), Figs. 11a-b.

329. Finally, Mills discloses that a single transcoder may transcode multiple portions of a video. Mills, ¶23-24. Accordingly, in some instances newly transcoded portions (e.g., *second segments*) would be stored in the same transcoded-content

store as a previously transcoded first portion (*storing the transcoded second segment in the local cache of the video optimizer*).

9. **Claims 8/18: The [method/system] of claim [7/17], further comprising [instructions that cause the processor to] [transmitting/transmit] to the database, instructions [to update/for updating] the database with a reference key corresponding to the transcoded second segment of the optimized version of the source file stored at the video optimizer.**

330. In my opinion, Mills combined with Mattis, with or without Sackstein, teaches or suggests this limitation for the reasons I have provided previously. §§X.C.3.b (2[b]/12[b]), X.C.8 (cl. 7).

10. **Claims 9/19: The [method/system] of claim [7/17], wherein transcoding the second segment of the original source file comprises: identifying a third segment of the optimized version of the original source file, subsequent to the second segment, that is stored by one of the plurality of other video optimizers; and transcoding the original source file from a start of the second segment to a start of the third segment.**

331. In my opinion, Mills, with or without Sackstein, teaches or suggests this limitation.

332. Mills, with or without Sackstein, provides multiple video transcoders capable of transcoding individual video portions upon request. §X.C.2.h (1[e]/11[g]). Additionally, both Mills and Sackstein check whether transcoded portions/segments are already stored and transcoding missing portions/segments. §X.C.8 (cl.7).

333. For these reasons, in my opinion, a POSITA would have understood that, in operation, when a portion of video has not been transcoded (*second segment*) is between two portions that have been (*first portion and third segment*), the portion that has not yet been transcoded would be (*transcoding the original source file from a start of the second segment to a start of the third segment*) to stream a complete transcoded video to the client. §§X.C.2.h (1[e]/11[g]), X.C.8 (cl. 7); X.C.1 (motivation). In fact, others that implemented the same features as Mills and Sackstein—including portioning multimedia for delivery—performed this exact feature. *See, e.g.*, Ex-1030, ¶¶42 (searching whether fragments already exist), 63 (“Unlike prepared media tracks which have all media fragments to form a complete alternative for the media, the additional alternatives may have none or only a subset of required media fragments prepared and the missing media fragments may be generated through transcoding of existing media fragments prepared for other alternatives when demanded”), Figs. 11a-b.

334. Finally, as I explained previously, because the combined system would provide multiple transcoders that may transcode different portions of the same video, a POSITA would have also understood based on Mills and/or Sackstein that (some) already-transcoded segments would be stored in different transcoded-content stores (*third segment...that is stored by one of the plurality of other video optimizers*). §§X.C.2.h (1[e]/11[g]), X.C.8 (cl. 7); X.C.1 (motivation).

11. Claims 10/20: The [method/system] of claim [1/11], wherein the first portion is maintained in the local cache of the video optimizer after one or more purges of the local cache.

335. In my opinion, Mills combined with Mattis, with or without Sackstein, teaches or suggests this limitation.

336. Mills discloses deleting a transcoded video if no longer necessary, but does not describe a detailed memory-management process for stored videos. Mills, ¶47.

337. Mattis, however, describes a “garbage collection” process to delete unnecessary objects (or fragments/portions) from cache. Mattis, 21:54-23:52.

338. In my opinion, a POSITA would have been motivated to modify Mills to implement a “garbage collection” process like Mattis’s to manage available memory in Mills’s transcoders (*video optimizers*). In doing so, a POSITA would have expected that a first portions/fragments of a video would be more-frequently accessed than later fragments. This is because it was well-known that end users would often start, but not complete playback of an entire video. Ex-1031, ¶34 (“many people will often start watching a video at its beginning, but will stop watching after a relatively brief period of time.”). Because Mattis implements a garbage-collection process that is fragment-specific and based on frequency of access, then, in many instances later fragments would be deleted (for being less frequently accessed) but the first portion would not. In fact, Sackstein describes a

similar deletion process to ensure sufficient cache memory remains available for newly transcoded portions of video, deleting old portions based on frequency of use and/or popularity. Sackstein, ¶¶45, 59.

339. Although Mattis notes that if one fragment of a multi-fragment object is marked for deletion, all fragments are similarly marked for deletion (Mattis, 23:24-31), in my opinion, a POSITA would have understood that Mattis did so because Mattis provided no way to re-create a deleted fragment, meaning deleting one fragment would likely corrupt the object. However, when providing transcoders capable of recreating deleted portions/fragments (like Mills does), a POSITA would have appreciated that there would be no need to delete all fragments if some subset are sufficiently active/popular and others are not, because inactive portions would simply be re-transcoded. This is again consistent with Sackstein, which envisioned deleting some, but not all, segments of a video based on popularity of individual segments. Sackstein, ¶¶45, 59.

XI. SECONDARY CONSIDERATIONS

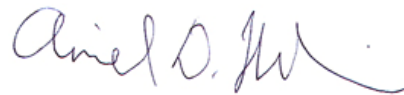
340. At this time, I am not aware of any secondary considerations of non-obviousness, or evidence sufficient to change my opinions that each of the Challenged Patents are obvious in view of the prior art. To the extent Patent Owner identifies any evidence of secondary considerations of non-obviousness, I reserve the right to respond to such evidence at that time.

XII. CONCLUSION

341. I, Aviel D. Rubin, declare that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: May 13, 2026

Respectfully submitted,

A handwritten signature in blue ink, appearing to read "Aviel D. Rubin", written over a horizontal line.

Aviel D. Rubin