A Collaborative Continuous Media Proxy Cache System

Kanela Kaligossi

Submitted to the Department of Electronic and Computer Engineering in Partial Fulfillment of the Requirements for the Diploma of Electronic and Computer Engineering at the Technical University of Crete.

Guidance Committee
Professor Peter Triantafillou (supervisor)
Associate Professor Manolis Koubarakis
Assistant Professor Euripides Petrakis

September 2002

Acknowledgements

I would like to thank Prof. Peter Triantafillou, for supervising this effort and for his guidance and support. I would like to thank Associate Prof. Manolis Koubarakis for participating in my jury and for accepting the typical role of the supervisor from June 2002, when Prof. Triantafillou left from Technical University of Crete. I also would like to thank Assistant Prof. Euripides Petrakis for participating in my jury and for his help regarding the process of preparing the presentation of this work.

Moreover I would like to thank Dimitris Michail for the very important discussions throughout the process of designing and implementing the system presented in this text. I also would like to thank Costas Harizakis for his willingness to participate in discussions that helped me solve various problems as well as for offering me his ideas regarding the monitoring of popularities of videos.

Abstract

The increasing popularity of media streams over the internet and the high start up delay that media steams typically experience, raised the urgent necessity of developing proxy caching systems dedicated exclusively for continues media.

The distinct characteristics of the streaming media and the users' access behavior suggest that the delivering and caching of streaming media must be handled in a different fashion than that of the traditional non-streaming objects such as HTML or image files. Techniques such as prefix caching become popular, in order to cache continuous media.

In this text, we present the design and implementation of a system for caching streaming media near to the clients, using their own resources. The users of the system form a hybrid peer-to-peer network and collaborate to become a large, aggregate, user-created cache.

Contents

Chapter 1

Introduction

1.1 Overview

The design of efficient systems for caching streaming media files is a challenging open research problem. With the augmented growing rate of the Internet and the wide availability of high-speed network access, an increasing number of streaming media objects are being distributed over the Internet. However, due to protocol overheads and poor delay, throughput and loss properties of the Internet, multimedia streams typically experience high start-up delay. One popular approach for reducing response time and network traffic is to cache popular streams at proxies near the requesting clients. In this work we present the design and implementation of a system for caching streaming media near to the clients using their own resources.

But is caching really useful? The answer would probably be yes if we considered the main reasons that made the caching technique so widely used. Firstly, since content is delivered from the proxy, which is closer to the client than the original server, the content is sent over a short network path, thus reducing the request response time, the total network resource usage and the probability of packets loss (improving quality perceived by the end–user). Furthermore, it helps prevent server from overloading, since the content can be delivered to users from a proxy and not from the original server.

In our system we have user machines forming a group and collaborate with each other to become a large, aggregate, user-created cache. In brief the system consists of a central process that undertakes the coordination of the clients and is responsible for making decisions such as, choosing the appropriate user where a stream should be cached or choosing the victim stream when replacement is needed. Users become part of the system by offering some space of their hard disk (actually this in not obligatory, a user may not offer resources if he is not willing to do so). They run a program that is capable, among others, of caching a stream in the hard disk, downloading a stream

from another client or from the distant Streaming Server and giving it to the player to play it out. Furthermore in order to reduce coordinator's load and reduce latency users inform each other about which titles are cached. In order to represent the cached titles we use "bloom filters" [16].

In the next three paragraphs we briefly present some key issues, concerning caching, streaming and peer-to-peer systems. Moreover, we will try to make clear the motivation and the objectives of this study.

1.2 Traditional Caching is Not Appropriate

Web caching has been used to accelerate the delivery of web objects such as HTML files and images. However streaming video objects differ from these web objects in a way that existing techniques for caching text and image object are not appropriate for caching media streams.

In particular, first of all, the size of video files is usually larger than non-streaming files by orders of magnitude and become larger as more network bandwidth becomes available. Moreover, video objects are mostly static content with the WORM (Write Once Read Many) property. User video access behavior and streaming media workload show different characteristics than those of non-streaming objects as reported in [17] and [18]. In particular users often view only the initial part of a video in order to determine if they are interested or not. Finally videos exhibit strong temporal locality. If a video has been accessed recently, chances are that it will be accessed again soon.

Given the large size of multimedia files and limited disk space at each proxy, storing the entire video file in a single proxy cache is therefore inefficient or even impossible. Hence, only a few hot video objects should be cached entirely. Most media objects should only be cached partially. In addition since users are likely to playback only part of the video, a proxy could cache a set of frames from the beginning of the stream. Therefore the proxy can serve a request by initiating transmission to the client from the cache while concurrently requesting the remainder of the stream from the server. This technique is called prefix caching [10].

1.3 Streaming Audio and Video on Demand

The technique, known as **streaming**, avoids having to download the entire audio/video before beginning playback. A client (player) begins playback of the multimedia stream a few seconds after it begins receiving the file from the player. In most applications a user may pause, rewind or fast-forward the multimedia content. Once playback of the multimedia begins, it should take place according to the original timing of the recording.

Upon a client request the streaming server partitions the file into segments which are typically encapsulated in RTP packets. Real-Time Transport Protocol (RTP) [15] provides end-to-end network transport functions suitable for applications transmitting audio or video. RTP is accompanied by RTCP a control protocol to allow monitoring of the data delivery. The need of a client/server interaction, for example pause/resume, led to the development of another protocol called Real–Time Streaming Protocol (RTSP) [19], which allows a player to control the transmission of a media stream (pause, fast-forward, rewind). RTSP messages use a different port number (usually 554) than the media stream.

An overview of the three protocols under discussion is presented in the next paragraphs.

1.3.1 Real-Time Transport Protocol (RTP)

RTP [15] provides the basic functionality for transferring real-time data over packet networks. Since RTP does not include mechanisms for reliable delivery or flow control, transport of RTP packets must rely on underlying protocols such as UDP and TCP. RTP typically runs over UDP, though TCP is sometimes used for reliable transport or to stream across firewalls that discard UDP packets. RTP provides source identification (randomly chosen SSRC identifier), payload type identification (to signal the appropriate decoding and playback information to the client), sequence numbering (for ordering packets and detecting losses), and timestamping (to control the playback time and measure jitter). Packets contain a generic RTP header with these fields, as well as payload-specific information to improve the quality of delivery

for specific media (e.g. MPEG). Interpretation of some header fields, such as timestamp, is payload dependent. The RTP header may also identify contributing sources (CSRCs) for the payload carried in the packet; such a list is typically inserted by mixers or translators.

1.3.2 Real-Time Control Protocol (RTCP)

RTCP [15] monitors the delivery of RTP packets. The protocol provides feedback on the quality of data distribution, establishes an identification (CNAME) for each participant, scales the control packet transmission with the number of participants (to avoid generating excessive feedback traffic in large multicast groups) and provides minimal session control information. RTCP packets consist of source descriptors, sender reports, receiver reports, BYE packets (signifying the end of participation in a session) and APP packets (for application-specific functions). Receiver reports include the stream's SSRC, the fraction of the lost RTP packets, the sequence number of the last RTP packet received and the packet interarrival jitter. Senders can use this information to modify their transmission rates or to switch to a different encoder. The sender reports include the stream's SSRC, the sequence number of the last RTP packet sent, the wallclock time of the last transmission and the number of RTP packets and bytes sent. The client can use the RTP timestamp and wallclock time information for media synchronization.

1.3.3 Real-Time Streaming Protocol (RTSP)

RTSP [19] coordinates the delivery of multimedia files and typically runs over TCP, although UDP can also be used. While conceptually similar to HTTP/1.1 [20], RTSP is stateful and has a number of different methods. The OPTIONS method inquires about server capabilities (e.g. RTP version number, supported methods, etc.) and the DESCRIBE method inquires about the properties of a particular file (e.g. Last-Modified time and session description information, typically using SDP [14]). Client and server state machines are created with the SETUP method, which also

negotiates transport parameters (e.g. RTP over unicast UDP on a particular port). The client sends a SETUP message for each stream (e.g. audio and video) in the file. Streaming of the file is initiated with the PLAY method, which can include a Range header to control the playback point. The TEARDOWN method terminates the session, releasing the resources at the server and client sites. The protocol also includes a number of recommended or optional methods.

1.4 Peer-to-Peer

The Internet as originally conceived in the late 1960s was a peer-to-peer system. The goal of the original ARPANET was to share computing resources around the U.S. The first few hosts on the ARPANET were already independent computing sites with equal status. The ARPANET connected them together not in a master/slave or client/server relationship, but rather as equal computing peers.

In recent years, the Internet has become more and more restricted to client/server type applications. But as peer-to-peer applications become common again, maybe Internet will revert to its initial design.

Peer-to-Peer systems seem to go hand-in-hand with decentralized systems. In practice, building fully decentralized systems can be difficult and many peer-to-peer applications take hybrid approaches to solving problems.

Napster is an example of a hybrid system. Its file sharing is decentralized. One Napster client downloads a file directly from another Napster client's machine, but the directory of files is centralized, with the Napster servers answering search queries and brokering client connections. This hybrid approach seems to work well when the number of peers is not very large. The directory can be made efficient and uses low bandwidth and the file sharing can happen on the edges of the network.

Other systems, such as Freenet have addressed decentralized control as a goal. All nodes are Peers and each Peer may function as router, client or server according to the status of the query.

Our system is a Peer-to-Peer system in the essence of having a large cache built out of clients' resources (clients' hard disk) and having clients directly contacting

each other in order to get a previously cached stream. The system follows the hybrid model with a central process acting as a control and coordination unit. Although the control is centralized we force clients do their utmost in order to avoid contacting central coordinator, by having them exchanging their cache directory (the list of URLs of cached streams) represented as a bloom filter. [16].

Chapter 2

System's Design Overview

In this chapter we present the general system architecture and its operation as well as the concepts and assumptions behind its design. Initially we outline the overall system and its constituent components. In the following step, we present how users exchange information about the cached content and describe the caching and replacement policies. Then we show how the system responds to user requests and finally we present how a user joins and leaves the system.

2.1 The Components of The System

Figure 1 shows an example configuration of the system. The distant Streaming Server is also present in the figure.

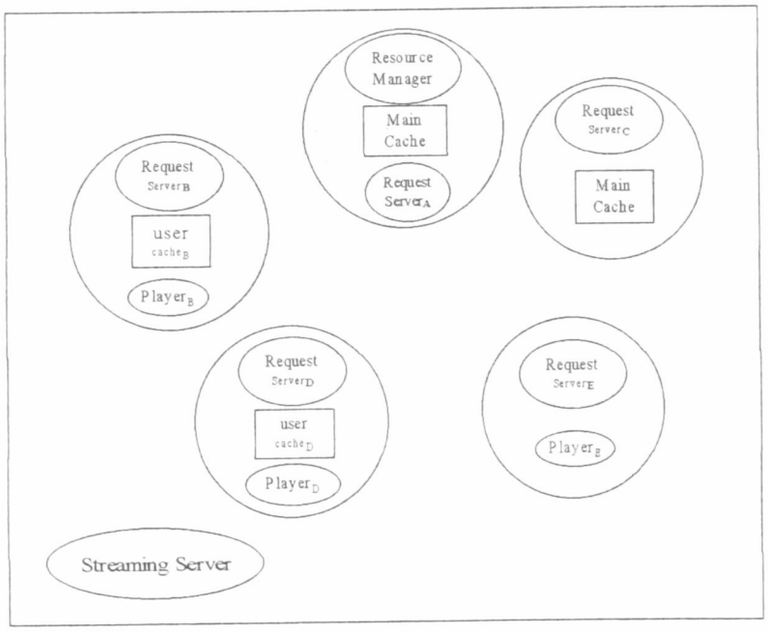

Figure 1: An example congiguration.

The components constituting the system are the following:

- A central process Resource Manager that is responsible for organizing the whole system. Resource Manager takes all the decisions concerning the managing of the data, e.g. running replacement algorithm, as well how users will collaborate.
- Request Servers which are processes running at users' machines. Request Servers are responsible to contact Resource Manager when a user has requested a video and get the necessary information to serve the request i.e. where the video is cached. Request Servers are also responsible for caching a video as well as give an already cached video for playback.
- User's cache, which is the amount of disk space that the user offered to the system and is where the videos are cached. A user may not offer disk space if he is not willing to.
- A large Main Cache, not necessarily located in one machine. This main cache guarantees that the system will offer the minimum of the services it is supposed to, even if no user offers disk space to the system.

In the following paragraphs we further comment on the components of the system, in order to clear out their role and their contribution to the operation of the whole system.

Resource Manager is responsible for all the decisions that are to be taken in the system. The decision, for example, of caching a video or not is taken by Resource Manager after having examined if the specific video is useful to be cached, that is if the caching of the video will probably increase the hit rate of the system. The decision of where to cache a video, if it is to be cached, is the result of the placement/replacement algorithm that Resource Manager runs upon a request for a video. Not every video will be cached in our system. In order to be cached, a title should satisfy some criteria set by the placement policy. Furthermore, although the machine that made the request for a video is generally preferable to be the place where the video will be cached, this is not always the case. If for example, this user

cache is full and there is another user cache with enough space to cache the specific video the video will be cached in the second user cache.

Request Servers are responsible for carrying out a user's request for a video. Upon a user request, Requests Servers either contact Resource Manager asking whether this video is cached and if they receive a positive response, where it is cached, or if they already know where to find it they contact directly the corresponding Request Server. A Request Server always waits for receiving messages either from Resource Manager with instructions, or from other Request Servers, asking it, for example to stream them a video that is cached in his user cache.

The large Main Cache that we decided to include in our architecture, although it guarantees that the minimum of the services will be offered, it has no special treatment. Thus we could say that it looks like an ordinary user cache with the only difference that it will "never" leave the system, at least not on purpose.

2.2 Advantages and Disadvantages of the Architecture

The above architecture provides us with a number of advantages. The most important advantage is the reduction in the latency faced by the user when a video is requested. This latency could be further reduced if the users where connected in a local area network. In this case the data transfers between the users can take advantage of the high bandwidth of a LAN.

By using users' hard disks we attain a high aggregate storage space, possible by orders of magnitude larger than that of a central proxy caching system.

Furthermore having the videos cached in different machines we distribute the load of serving video requests over several machines. This is a better approach than having a central proxy that services all the video requests and becomes a system bottleneck.

The main problem of this architecture is that Resource Manager is a central point of failure. Despite this fact, we preferred this approach by having in mind, that searching for content in a pure peer-to-peer system requires an often very expensive

distributed search. Moreover, building fault tolerant central servers, is a well studied problem.

2.3 Exchanging Summaries of Cached Titles

In order to reduce the load of Resource Manager, and avoid that every request for a video will pass through it as well as to save bandwidth resources, we adopt the approach proposed in [21]. Every Request Server maintains a summary of the cached content. In order to reduce memory requirements, we store each summary as a "Bloom filter" [16]. Request Servers periodically exchange their summaries and every Request Server keeps a bloom filter for each user cache describing the cached content. Thus when a user requests a video that is not cached in its machine Request Server checks all the bloom filters in order to identify if the video is cached somewhere else and if it is, it directly contacts the corresponding Request Server. By doing so, it is not necessary for all requests to pass through the Resource Manager.

In order to avoid that a user contacts every body else when he wants to send his summary we have users connected in a tree topology as shown in figure 2.

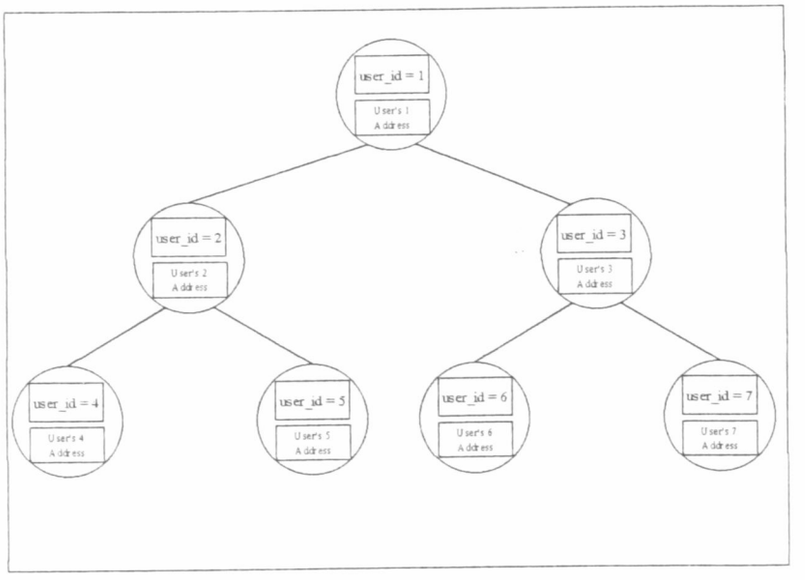

Figure 2: The tree topology.

Thus a user sends his summary cache only to his neighbors (parent and children) and the summaries traverse up and down the tree and consequently every body is very quickly informed about an update of a bloom filter.

The relevant algorithm works as follows:

- Requests servers maintain a bloom filter for every user in the system.
- Request Servers send their updated bloom filters, either periodically or after a certain percentage of change in the contents of the cache. When a Request Server wants to send an updated bloom filter, prepares the message and sends it to his parent as well as to his children in the tree topology.
- If a Request Server receives a message with an updated bloom filter, updates the data structure that keeps the bloom filters and checks if the message came from its parent or from a child.
- If the message came from the parent, it forwards it to both of its children.
- If the message came from a child, it forwards it to the parent as well as to the other child.

2.4 Cache Admission Policy and Replacement Policy

Research has shown that the popularity distribution of the videos, follows Zipf's low [18]. In other words we could say that about 60% of requests are for a small number of titles (about ten of them). This fact led us to the decision of caching these very popular titles in their entirety. This is feasible since the number of those titles is small. This simple decision will increase a lot the hit rate of the system.

Furthermore previous research has shown that the performance of a system that caches videos is improved by having cached as many prefixes of videos as possible.

The placement and replacement of the videos take place by having the previous goals in mind as well as by trying to keep the system load balanced. If for example we decide that a very popular video should be cached, we choose the machine with the least load as the place where the video will be cached. We represent the load of a

machine by the sum $\sum_{i=1}^{k} p_i$ where p_i represents the popularity of video i, and k is the number of titles cached in the corresponding machine.

At the procedure of caching a video we give the highest priority to the N most popular videos. The next priority is given to the prefixes and if there is enough space left we cache suffixes of the videos with popularities that have the next highest values after the popularities of the N most popular videos. Therefore prefixes can only be replaced by a videos belonging to the N most popular ones. Suffixes are replaced either by prefixes or by a video belonging in the N most popular category.

2.4.1 Prefix Size

The size of the video prefix that should be cached depends on items such as round-trip delay, server-to-proxy latency, video specific parameters (size, bit rate) and transmission rate of lost packets. As paper [10] suggests, suppose that the delay from the server to the proxy ranges from d_{min} to d_{max}. To support discontinuity-free playback with a start-up delay of s to the client, we store a prefix of at least $\max\{d_{max} - s, 0\}$ frames.

2.4.2 Monitoring Popularities of the Titles

Request Servers keep track of the hits of every title and periodically send to Resource Manager a message with the number of hits of every title. Resource Manager keeps a sorted list of the titles in decreasing order based on their hits. Since we are not interested in the exact popularities of the videos but rather in the relative difference between popularities, Resource Manager uses a Zipf distribution with constant theta in order to represent videos' popularities. The distribution becomes more skewed as the theta is increased. A theta for example of value 0.7 would help us separate the most popular videos than the least ones. Resource Manager sets a cutoff threshold in Zipf and the titles that have popularity beyond this threshold are not eligible to be cached.

2.4.3 N Most Popular Videos

As we have already mentioned we have decided that the videos, to which the greatest amount of requests refer, should be cached in their entirety. We refer to those videos as the N most popular ones. In order to decide which of the titles belong to the N most popular category, we choose a percentage **a** let us say, 60% and we take the popularities of the first titles in Zipf and add up to this percentage. The N most popular titles will be those that contributed to the summation.

2.4.4 Placement and Replacement Algorithms

Upon a user request for a title that is not already cached in the system, Resource Manager runs placement and replacement algorithms. The placement algorithm decides if the video should be cached and if it is yes, then if there is not enough space to be cached, replacement algorithm chooses the victim stream.

2.5 Demonstration of some Protocols

In this section we describe how the system reacts to user requests. We use two common scenarios, a cache miss which means that the requested video is not cached in the system and a cache hit meaning that the requested video is already cached.

The following events occur when $Player_A$ asks Request $Server_A$ for video V.

2.5.1 Cache Miss

Let us assume that the video V is not cached.

- Request $Server_A$ checks its bloom filter, as well as the bloom filters that keeps for the other users, to find out if this title is cached somewhere. It does not find any information so,
- It contacts Resource Manager requesting the specified video. Resource Manager replies negatively. It also runs placement and replacement algorithms and gives instructions on what to do with this new title. These instructions may say that Request $Server_A$ should cache the video (possible by replacing an other video), or should give it to another Request Server to cache it in its cache or do nothing more than give it to $Player_A$.

- So Request Server$_A$ requests the video from the Streaming Server and begins downloading giving the video to Player$_A$, as well as follow the instruction that he took from Resource Manager.

2.5.2 Cache Hit

Let us assume now that video V is cached in user cache$_B$. Player$_A$ asks Request Server$_A$ video V.

- Request Server$_A$ checks its bloom filter, as well as the bloom filters that keeps for the other users, to find out if this title is cached somewhere.
- It finds out the video is cached in user cache$_B$. (The case may be that the prefix is in user cache$_B$ and the suffix in user cache$_C$.) In either case it contacts the corresponding Request Servers and gets the video.
- If the suffix is not cached, after getting the prefix it contacts the Streaming Server and gets the suffix.
- If it does not know where the video is cached, it contacts Resource Manager requesting the specified video.
- Resource Manager replies in affirmative and informs it that the prefix of video is cached in user cache$_B$ (and the suffix let's say in user cache$_C$).
- Request Server$_A$ contacts Request Server$_B$ and gets the prefix (and then contacts Request Server$_C$ and gets the rest of the video). If the video title is partially cached Request Server$_A$ receives the suffix from the Streaming Server.

2.6 Joining the System

A user that would like to join the system should firstly contact Resource Manager declaring his intention to be part of the system and giving information about the resources that he would like to offer. The resources concern the amount of disk space that he would like to offer which will be managed by Resource Manager.

After a new user is encountered in the system, Resource Manager updates its information concerning the state of the system and sends to the new user the bloom filters of the users that already participate to the system, as well as their addresses. It

also informs all the users about the arrival of the new one giving them his address. The users, including the new one, update their data structures and find their new neighbors in the tree topology.

2.7 Leaving the System

Users will not be part of the system "for ever ". Hence a user may leave the system, either because his/her machine crashed, or because he/she does not want any more to participate in the system. Two mechanisms are used in order to handle the users' departures. *Intention to Leave* is used when the user is willing to leave after informing about his intention to leave and hence he leaves the system in valid state. And *Leave Immediately* is used when user's machine crashes or the power button is simply pushed.

2.7.1 Intention to Leave

In this case Request Server of the corresponding user machine contacts Resource Manager reporting its intention to leave. However, if it is already streaming a video to another user, the user should wait until the transfer is complete.

- Resource Manager checks the record of that user.
  - If there is an important title in that machine, it contacts Request Server of another machine requesting it to download the specified video from the machine that is going to leave the system.
  - If it is already in charge of downloading data, which are still useful, it must wait until downloading is completed and data are taken from that machine.
  - Then the user is free to leave.
- Resource Manager updates its data structures and sends a message to everybody else that the specific user left the system. The remaining Request Servers update their data structures, including the removal of the bloom filter that corresponds to the user that left the system and finding their new neighbors in the tree topology used to exchange the bloom filters.

2.7.2 Leave Immediately

In this case the user leaves the system with out any previous warning. The other users can notify his absence in three ways. So let us assume that $user_A$ suddenly leaves the system. Then $user_B$ can detect his absence in the following ways:

- $User_B$ knows that the video that he wants is cached in $user_A$. He contacts $user_A$ and gets no response.
- $User_B$ gets data from $user_A$ who suddenly left the system. Then $user_B$ just does not receive anything anymore.
- $User_B$ is streaming a video to $user_A$ and suddenly he can not send him anymore data.
- $User_B$ fails to send a message with updated summaries to $user_A$.

In all the above cases, $user_B$ informs Resource Manager about the failure. Then Resource Manager after confirming that $user_A$ is no more part of the system updates its data structures and informs the remaining users about the departure of the specific user. They also update their data structures, including the removal of the bloom filter that corresponds to the user that left the system and finding their new neighbors in the tree topology.

Chapter 3

Implementation

The completion of our work includes the implementation of an initial prototype that implements most of the main features of our architecture along with the basic modifications necessary for interacting with the Apple's streaming server named "Darwing Streaming" Server and Apple's multimedia player, "QuickTime Player". Using the basic implementation, we can cache a stream partially or in its entirety, and we can successfully process client RTSP messages, requesting the full contents from the server on a cache miss, or partial contents when a portion of the stream resides in the cache. Moreover, clients can stream a previously cached video, one to each other, and can find the location of a cached stream not only by requesting the corresponding URL form Resource Manager but also through the mechanism of exchanging their bloom filters.

3.1 Request Server

The Request Server process behaves as a media server to the clients and as a media client to the servers. It also manages its local cache, collaborates with other Requests Servers and communicates with the Resource Manager process either for asking the location of a video or for getting instructions on what to do with a specific title. Every client has a unique user id and the Request Servers use this user id in order to address each other, as well as to be addressed by the Resource Manager.

3.2 Resource Manager

The Resource Manager process is responsible for running the placement and replacement algorithms, as well as for coordinating the behavior of the Request Servers. Resource Manager always waits for messages coming from the Request Servers. A Request Server opens a TCP connection in order to contact Resource

Manager and sends a message. Resource Manager replies using the same TCP connection.

3.3 RTSP/RTP client and server

Unlike HTTP, the control protocol RTSP, used for media streaming is not a stateless protocol. The state for media streamed to the clients by the Request Server is managed by the Request Server. For example the session identifier will be issued and managed by the Request Server for the sessions for which the Request Server is the data source. This does not preclude the actual server from managing a session state since Request Servers might go directly to the server for a non cached video.

In the next paragraphs we will discuss about the sequence of RTSP messages exchanged between a client (player), a Request Server and the streaming server. An example is shown in figure 3 where the prefix is streamed from the cache and the suffix is requested from the Streaming Server. In Appendix A the reader can find a real RTSP session.

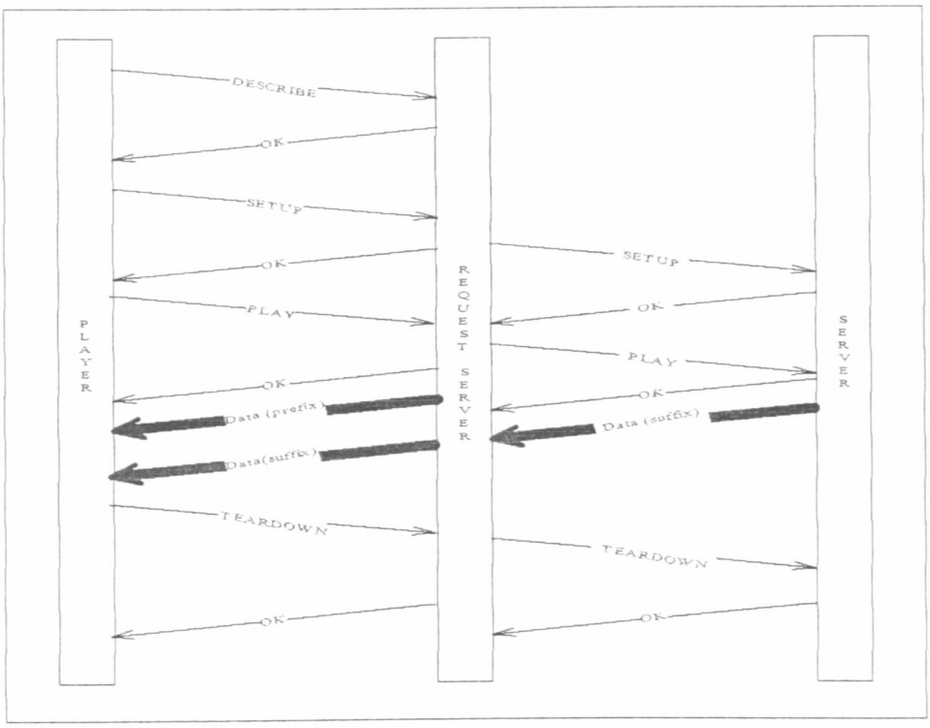

Figure 3: RTSP messages exchange under prefix caching.

The handling of client requests depends on whether or not the requested video is cached (partially or in its entirety).

The client begins requesting a video by sending a DESCRIBE message to Request Server. With this message, the client requests information about the particular video, which is usually described with the Session Description Protocol (SDP), RFC 2327 [14]. If the video resides in the local cache or in the cache of another machine, its description is also cached and the proxy responds directly to the client either by reading the description from the local cache or by requesting it from the Request Server where the video is cached. If the video is not cached Request Server contacts the streaming server and forwards the client's DESCRIBE message and forwards the server's response to the client.

The SETUP request is used to negotiate the transport parameters, including the transport protocol (e.g. RTP) and the TCP/UDP port numbers. Upon receiving the client SETUP message, port numbers are generated for the Request Server's end of the RTP and RTCP connections, and a session identifier is selected. The port numbers and session identifier are sent to the client. If the video is cached to another's machine cache, Request Server sends the SETUP message to the Request Server of the specific machine, who gets prepared in order to stream the video. If the video is not cached, Request Server generates a separate SETUP message which is being sent to the streaming server. Each stream in the multimedia presentation results in separate connections on both the Request Server – client and streaming server – Request Server paths and it consists of UDP connections for RTP and RTCP. To coordinate the transfer of RTP and RTCP messages, the Request Server maintains a mapping table to direct messages to the appropriate outgoing connection (and with the appropriate session identifier).

The client sends the PLAY message after receiving the Request Server's response to the SETUP request. If the video is cached in the local cache the Request Server responds immediately to the PLAY request and initiates the streaming of RTP messages to the client. If the video is cached in an other machine the Request Server streams the video to the client, by receiving it from the Request Server of the machine where the video is cached. If the video is not cached the Request Server forwards the

PLAY request to the streaming server that starts the streaming of the video and which is forwarded to the client. If only a prefix is cached, the Request Server while streaming the prefix prepares a SETUP and PLAY request to send to the streaming server, with the appropriate Range header and asks the suffix from the streaming server.

3.3.1 Linking the Prefix and Suffix

When the entire stream resides in the cache, the Request Server acts as a server in responding to the client PLAY request. When the cache does not contain any portion of the requested video, the Request Server forwards the PLAY request to the server and simply forwards RTP and RTCP packets from the server to the client and RTCP packets from the client to the server acting as an application level router. The operation of the Request Server becomes more interesting when the cache stores only the prefix of the stream. In this case, the Request Server replies to the client PLAY request and initiates transmission of RTP packets to the client and when prefix is streamed, it requests the suffix from the server. Fetching the suffix requires the Request Server to send a PLAY request to the server with the appropriate Range header in order to ask the appropriate portion of the stream.

As part of caching multimedia content, the Request Server keeps track of the size of the prefix in terms of the timestamps in the RTP packets. The RTSP Range Request is defined for both SMPTE Relative timestamps and NPT. SMPTE has the format hours:minutes:seconds:frames.subframes with the origin at the beginning of the clip. In contrast, NPT time consists of a decimal fraction, where the left part of the decimal may be expressed in either hours, minutes or seconds, and the right part of the decimal point measures fractions of a second. These formats require the Request Server to convert the RTP timestamps into time, by applying the payload-specific clock rate [15].

In order to link the prefix and the suffix, all RTP headers must be consistent, otherwise the client will not be able to associate the two parts to the same stream. The sensitive fields are sequence numbers, timestamps, and source identifier (SSRC), which have been selected separately by the Request Server (for the prefix) and the

server (for the suffix). In streaming the suffix, the Request Server overwrites the SSRC field in each RTP packet with the value he has selected as part of initiating transmission of the prefix. The Request Server knows the timestamp and sequence number used in transmitting the last RTR packet of the prefix. The base timestamp and sequence number for the server's transmission of the suffix are provided in the RTP-Info header in the PLAY response. The Request Server can then add/subtract the appropriate constant for the timestamp and the sequence number fields of each packet in the suffix.

3.4 Caching a Video

When a user joins the system, he offers some specific amount of the disk space, which will be used for caching videos. This space is actually a specific directory in the user's filesystem. It may for example be "C:\foo\bar\cache". This directory is the one that the Request Server will use in order to cache the various videos.

The caching of a video includes the creation of a directory (inside cache directory) with a name that is a result of the MD5 signature of the video's URL. An ordinary multimedia stream is consisted of two streams, one for the audio stream and one for the video stream. These streams are cached inside the directory that corresponds to the video, as files with a name that is the MD5 signature of the stream's URL. Moreover, the name of a stream has as a prefix a string that denotes if this file is the prefix or a suffix of the stream. Inside the video's directory the Request Server also caches metadata information about the video, i.e. the response to the DESCRIBE request. Thus to give an example, an ordinary directory that corresponds to a video with URL:

rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov

and with URLs for the audio and video tracks:

rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov/trackID=3,

rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov/trackID=4 respectively,

would be of the form:

- Directory's name: 10daadc274e484c8bbdd5263ef01a222

- Files contained inside the directory:
  - Describe
  - prefix08a36bfee71a5b5d60e6000fd9e522cd
  - suffix08a36bfee71a5b5d60e6000fd9e522cd
  - prefixc5d547eb9e2f2b22b8bcdf1df6eca331
  - suffixc5d547eb9e2f2b22b8bcdf1df6eca331

The Request Server has specific procedures in order to map a URL to a filename and vice-versa. In the previous example of a video directory we saw that the video is cached in its entirety, since we have both the prefix and the suffix for the audio/video streams. If only the prefix of a video is cached then the corresponding directory would only have two files named for example "prefix08a36bfee71a5b5d60e6000fd9e522cd" and "prefixc5d547eb9e2f2b22b8bcdf1df6eca331" that correspond to the prefix of the audio/video stream as well as the "describe" file with the metadata information.

Our goal was to make the system independent of the media encoding mechanism. Hence we cache RTP packets rather than raw multimedia content. This obviates the need to parse the body of the RTP packets and repacketize the content, at the expense of additional storage overhead for RTP headers. Since packets sometimes arrive out of order, we do not write to the disk, a packet as soon as it arrives, trying not to discard some late packets. Instead we maintain a list of RTP packets in the main memory, sorted by the packet's sequence number and we insert a late packet in the appropriate place in the list. When the list's capacity is over, we write the packets to the disk.

3.5 Bloom Filters and Tree Topology

The Request Server maintains a data structure for keeping the bloom filter that corresponds to the local cache directory as well as a data structure for keeping the bloom filters of the other clients. The Request server also keeps the tree topology used to exchange the bloom filters. The nodes of the tree contain a mapping from a user id to the network address of the user's machine.

The tree topology was maintained using the TreeMap class of Java, which is a Red-Black tree based implementation. Thus it guarantees that the map will be in

ascending key order. The keys of the mapping are the user ids. The implementation provides guaranteed log(n) time cost for the searching, inserting and removing operations.

When a new user is encountered in the system, or a user leaves the system Resource Manager informs all the Request Servers for this change. The Request Servers rearrange the tree topology to be consistent with the new structure of the system and then find their new neighbors (parent and children) in the tree topology.

The exchanging of the bloom filters takes place according to the procedure described in section 2.3

Chapter 4

Related Work

In the recent years, much work has been done on the subject of streaming multimedia. Some recent work has focused on distributed media servers [5,2,8,9,12]. Furthermore, lot of techniques have been developed in order to improve the video quality perceived by the end user, and to reduce the load faced by the streaming servers. Among others, these techniques include caching, preffetching, batching and multicasting. We present here some of the work done on those techniques.

4.1 MiddleMan

The work closest related to ours is that of Acharya and Smith in MiddleMan [1]. They introduce the concept of having clients to act cooperatively in order to cache video files. A video is partitioned in a number of equal blocks and caching takes place on a block by block basis. The numerous blocks of a video may be cached in different client machines. And in order to serve a request for a title, the process that handles the request, sequentially asks the coordinator of the system for the location of every block of the video.

Middleman suffers a lot from wasting bandwidth resources due to the big amount of messages that have to be exchanged between clients and the coordinator of the system. The messages exchanged, in order for a client to get a video, are in the order of O(3n), where n is the number of blocks that the video is segmented. This amount of messages can not be tolerated, therefore in our work we concentrated on having a small and constant number of messages exchanged between clients and Resource Manager in the procedure of getting a video.

Moreover, the partitioning of the video in numerous blocks and the caching of each block in a different machine, introduces serious synchronization problems. MiddleMan does not provide any mechanism in order to solve those synchronization problems. We would dare to say that the design of Middleman is not (if we can use

such a term) "*implementation oriented*". In our opinion the implementation of a system based on Middleman's design it is highly unlikely that it would ever work as it was supposed to.

4.2 Caching in Centralized Proxy Servers

Other recent work has focused on having cached a fixed amount of frames of a video in a centralized proxy server. Towards this goal, a lot of techniques have been proposed, some of which are presented in the following paragraphs. We could say that they are various aspects of the prefix caching technique and would be useful to any system that caches videos.

Addressing the problem of high latency and loss rates in the Internet, [10] proposed a prefix caching technique whereby a proxy stores the initial frames of popular clips. Storing part of each audio/video stream enables the proxy to reduce client delay without sacrificing quality. Upon receiving a client request, the proxy initiates transmission to the client from the prefix cache, while simultaneously it requests the remainder of the frames from the server. The proxy can reduce the network resource requirements by performing workahead smoothing into the client playback buffer. Transmitting large frames in advance of each burst can substantially reduce the peak rate and variability of the video stream. A detailed model of workahead smoothing in the presence of a prefix buffer, is presented and it is showed how to compute smoothed transmission schedules. They also introduce techniques for allocating buffer and bandwidth resources across multiple clients, based on the popularity and resource requirements of each stream. To further reduce the buffer requirements, they describe how to multiplex access to a shared smoothing buffer across multiple streams.

Another recent work has presented a segment-based buffer management approach to proxy caching of media streams [13]. In this approach, blocks of a media stream received by a proxy server are grouped into variable-sized segments. The cache admission and replacement policies then attach different caching values to different segments, taking into account the segment distance from the start of the media. These

caching policies give preferential treatments to the beginning segments. This approach, although best applied to a centralized proxy server can be further extended to work on a proxy system consisted of caches spread among several clients, as is our system.

In [6] was suggested that the initial frames of a video should be cached in order to prevent a jitter playback. They also propose that some additional frames should be cached, if more space left in the cache. The two approaches are "additional initial caching" and "selective caching". In the second approach, selective caching, selects the frames to be cached based on the knowledge of user buffer and video stream properties. It tries to give the maximum benefit to the user, in terms of increasing the robustness of entire video stream against network congestion, while not violating the user buffer size limit.

A network-conscious approach to the problem of end-to-end video delivery over wide-area networks using proxy servers situated between local-area networks and a backbone wide-area network was presented in [11]. The major objective of this approach is to reduce the bandwidth requirement in the backbone wide-area network. Their basic idea is to prefetch a predetermined amount of video data and store them a priori at proxy servers. This operation is referred as staging. They focus on developing video staging methods. Based on these methods, they developed heuristic algorithms to solve the problem of determining the amount of video data to be staged at the proxy server, so as to minimize the total backbone bandwidth requirement.

Techniques for improving cost/performance for on-demand video delivery including multicast data delivery, segmented data delivery and proxy servers are presented in [4]. They show how the dynamic skyscraper technique [8] can be modified such that proxy servers can cache just the first few segments of an object. They also developed an optimization model that can be solved to determine the form of the optimal proxy caching strategy in a system with homogeneous proxy caches. The model computes the cache content that minimizes delivery cost, constrained by the storage capacity and bandwidth available at the proxy servers.

The implications of congestion control and quality adaption on proxy caching mechanisms are addressed in [7]. They investigate the use of layered encoding, in

order to adjust the quality of the playback of a multimedia stream. With the layered codec, the compressed data is split into a base layer which contains the most essential low quality information and other layers which provide optional enhancement information. Thus, the more layers are played back, the better the quality becomes. They present a fine-grain replacement algorithm for layered-encoded multimedia streams at Internet proxy servers, and describe a pre-fetching scheme to smooth out the variations in quality of a cached stream during subsequent playbacks. They also extend the semantics of popularity and introduce the idea of weighted hit, to capture both the level of interest and usefulness of a layer for a cached stream.

4.3 SpreadIt

A recent work has focused on finding a way to stream live media by using client resources in order to reduce server's load and to overcome the problem of the limited server's resources that do not permit the serving of an indefinite increasing number of requests. This architecture called *SpreadIt* presented in [3] suggests the streaming of live media over a network of clients. SpreadIt builds an application level multicast tree, over a set of clients which form a peer-to-peer network. Each client node is enabled with a basic *peering layer* between the application and transport layers. A node can function as server and a client simultaneously. As a client, it receives the stream from some node in the network. As a server, it forwards the stream to other nodes in the network. The player gets the stream from the peering layer on its local machine. The peering layers at different nodes coordinate among themselves to establish and maintain a multicast tree. Since the clients are autonomous and subscribes and unsubscribes are unpredictable, the peering layer is responsible for hiding changes in the multicast topology from the applications above.

Chapter 5

Conclusions and Future Work

The increasing popularity of media streams over the Internet and the high start up delay that media streams experience, raised the urgent necessity of developing proxy caching systems dedicated exclusively for continuous media.

In these work, we presented the design and implementation issues of a system for caching media streams. By trying to take advantage of previously unused resources (user disk space), we developed an architecture whereby the cache of the system is distributed over the client machines, that collaborate in order to cache and stream the video objects. The collaboration takes place with the help of a central unit that coordinates the system. We also tried to reduce the load of the central process and the response time by having the users to exchange their cache directory represented as a bloom filter. Therefore, not every request for a video has to pass through the central process, reducing in that way its load. Moreover, by avoiding to contact the coordinator for every request, the response time and latency faced by the user are reduced.

Throughout the procedure of designing and implementing the system we realized that in a system whereby the cache is distributed, there is a tradeoff between efficient managing of the cache space and the quality perceived by the end user. Due to the large size of the video files, their partitioning in segments that are cached where ever there is available space would lead to a more efficient cache management. But on the other hand this partitioning leads to serious synchronization problems, even to the total disruption of the playback procedure.

Regarding the implementation of the system, the most of the difficulties we faced, concerned the streaming part of the system. Fetching a video from a streaming server, stream a video from the cache to the player and moreover stream a video from an other's machine cache to the local player, was not a simple task. These difficulties

may be the main reason, that until the moment that this text is written, there is not any commercial or not commercial product for caching multimedia streams.

Moreover, and this may sound ordinary, we faced the problem of the "enormous distance between theory and practice". Much work has been done on caching video streams in theory but few of these ideas look like implementable.

As a future work we intend to realize a simulation and gather experimental results regarding the hit rate of the system, the latency and response time and other parameters that characterize proxy caching systems.

Moreover, we intend to design more efficient placement and replacement algorithms that better fit to the distributed nature of the cache space.

Finally to find mechanisms that better support the unpredictable subscribes and unsubscribes of the peers.

Appendix A
An RTSP Session

DESCRIBE rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov RTSP/1.0
Cseq: 1
Bandwidth: 56000
Accept-Language: en-US
Accept: application/sdp
User-Agent: QTS (qtver=5.0.2;os=Windows 98 A)

RTSP/1.0 200 OK
Content-Type: application/sdp
Expires: Sat, 24 Aug 2002 15:07:46 GMT
Date: Sat, 24 Aug 2002 15:07:46 GMT
Server: DSS/4.0 [v410]-Linux
Cseq: 1
Content-Length: 338
X-Accept-Retransmit: our-retransmit
Content-Base: rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov/
Last-Modified: Sun, 26 May 2002 15:16:19 GMT
Cache-Control: must-revalidate

v=0
o=StreamingServer 3239190466 1022426179000 IN IP4 147.27.7.101
s=/sample_56kbit.mov
u=http:///
e=admin@
c=IN IP4 0.0.0.0
t=0 0
a=control:*
b=AS:33
a=range:npt=0- 59.76000
m=audio 0 RTP/AVP 96
b=AS:14
a=rtpmap:96 X-QT/22050/1
a=control:trackID=3
m=video 0 RTP/AVP 97
b=AS:19
a=rtpmap:97 X-QT/600
a=control:trackID=4

SETUP rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov/trackID=3 RTSP/1.0
Cseq: 2
Transport: RTP/AVP;unicast;client_port=6970-6971
X-Transport-Options: late-tolerance=1.500000
X-Retransmit: our-retransmit
Accept-Language: en-US
User-Agent: QTS (qtver=5.0.2;os=Windows 98 A)

RTSP/1.0 200 OK
X-Transport-Options: late-tolerance=1.500000
Expires: Sat, 24 Aug 2002 15:07:46 GMT
X-Retransmit: our-retransmit
Date: Sat, 24 Aug 2002 15:07:46 GMT
Server: DSS/4.0 [v410]-Linux
Transport: RTP/AVP;unicast;client_port=8000-
8001;source=147.27.7.101;server_port=6970-6971;ssrc=4894AF31
Cseq: 2
Session: 8421241561181371392
Last-Modified: Sun, 26 May 2002 15:16:19 GMT
Cache-Control: must-revalidate

SETUP rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov/trackID=4 RTSP/1.0
Cseq: 3
Transport: RTP/AVP;unicast;client_port=6972-6973
X-Transport-Options: late-tolerance=1.500000
Session: 8421241561181371392
X-Retransmit: our-retransmit
Accept-Language: en-US
User-Agent: QTS (qtver=5.0.2;os=Windows 98 A)

RTSP/1.0 200 OK
X-Transport-Options: late-tolerance=1.500000
Expires: Sat, 24 Aug 2002 15:07:46 GMT
X-Retransmit: our-retransmit
Date: Sat, 24 Aug 2002 15:07:46 GMT
Server: DSS/4.0 [v410]-Linux
Transport: RTP/AVP;unicast;client_port=8002-
8003;source=147.27.7.101;server_port=6970-6971;ssrc=47F9C8E0
Cseq: 3
Session: 8421241561181371392
Last-Modified: Sun, 26 May 2002 15:16:19 GMT
Cache-Control: must-revalidate

PLAY rtsp://panoramix.softnet.tuc.gr/sample_56kbit.mov RTSP/1.0
Cseq: 4
Range: npt=0.000000-59.760000
Session: 8421241561181371392
X-Prebuffer: maxtime=2.000000
User-Agent: QTS (qtver=5.0.2;os=Windows 98 A)

RTSP/1.0 200 OK
Cseq: 4
Server: DSS/4.0 [v410]-Linux
Session: 8421241561181371392
Rtp-Info:
url=trackID=3;seq=7245;rtptime=529540800,url=trackID=4;seq=64484;rtptime=143
7627664

References

[1] S. Acharya and Brian Smith. Middleman: A video caching proxy server. *Proceedings of the NOSSDAV 2000, Chapel Hill, NC,* June 2000.

[2] Stergios V. Anastasiadis, Kenneth C. Sevcik, and Michael Stumm. Modular and efficient resource management in the EXEDRA media server. *In 3rd USNIX Symposium on Internet Technologies and Systems, San Francisco, California,* March 2001.

[3] Hrishikesh Deshpande, Mayank Bawa, and Hector Carcia-Molina. Streaming live media over a peer-to-peer network.

[4] Derek L. Eager, Michael C. Ferris, and Mary K. Vernon. Optimized regional caching for on-demand data delivery. *In Proc. Of Multimedia Computing and Networking*, Jan 1999.

[5] Zihui Ge, Ping Ji, and Prashant Shnoy. A demand adaptive and locality aware (DALA) streaming media server cluster architecture.

[6] Zhourong Miao and Antonio Ortega. Proxy Caching for Efficient Video services over the internet. *In Ninth International Packet Video Workshop (PVW'99), New York,* April 1999.

[7] R. Rejaie, M. Handley, H. Yu, and D. Estrin. Proxy caching mechanism for multimedia playback streams in the internet. *4th Web Cache Workshop, San Diego, CA.*, March 1999.

[8] Olav Sandsta, Stein Langorgen, and Roger Midtstraum. Video server on an atm connected cluster of workstations. *In International Conference of the Chilean Computer Sciecnce Society,* pages 207-217, 1997.

[9] J.R. Santos and R.Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. *In 6th ACM International Multimedia Conference, Bristol, United Kingdom,* Sep. 1998.

[10] Subhabrata Sen, Jennifer Rexford, and Donald F. Towsley. Proxy Prefix Caching for Multimedia Streams. *Proc. IEEE INFOCOM,* March 1999

[11] Yuewei Wang, Zhi-Li Zhang, David Hung-Chang Du, and Dongli Su. A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers. *In Proc. Of IEEE INFOCOM,*pages 660-667, April 1998

[12] W.Bolosky, J. Draves, R.Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, and R. Rashid. The Tiger video fileserver. *In NOSSDAV '96,* April 1996.

[13] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf, Segment-based proxy caching of multimedia streams. In *World Wide Web,* pages 36-44, 2001.

[14] M. Handley and V. Jacobson, "SDP: Session Description Protocol," Request for Comments (Proposed Standard) RFC 2327, Internet Engineering Task Force, Apr. 1998.

[15] H.Schulzrine, "RTP profile for audio and video conferences with minimal control", February 1999. Internet Draft ietf-avt-profile-new-05.txt, work in progress.

[16] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM,* pages 13(7):422-426, July 1970.

[17] S. Acharya, B. Smith, and P.Parnes, "Characterizing User Access To Videos On the World Wide Web", *Proceedings of the SPIE/ACM MMCN 2000,* San Jose,CA, January 2000, pages 130-141.

[18] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy, "Measurement and Analysis of a Streaming-Media Workload", *Proceedings of the USITS'01*, San Francisco,CA, March 2001.

[19] H. Schulzrine, A. Rao, and R.Lanphier, "Real time streaming protocol (RTSP)", *Request for Comments 2326,* April 1998.

[20] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1", *Request for Comments 2616,* June 1999.

[21] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol" *"IEEE/ ACM Transactions on Networking"* , 2000