Using Network Flow Buffering to Improve Performance of Video over HTTP

Jesse Steinberg and Joseph Pasquale
University of California, San Diego
Department of Computer Science and Engineering
La Jolla, CA 92093 USA
(1) 858-534-8604
{jsteinbe,pasquale}@cs.ucsd.edu

ABSTRACT

We introduce the notion of *network flow buffering* for direct HTTP streaming of web-based video clips, making it a more viable alternative to specialized streaming protocols, and in some cases making it a superior approach. A network flow buffer is simply a remote buffer that is dynamically deployed between a Web client and server, and that actively or passively regulates the flow of streaming video data. Network flow buffering is a simple approach to improving the performance of Web-based streaming video by buffering at a location where it can operate more effectively, providing benefits such as supplementing client buffering capacity, smoothing differences between WAN and LAN bandwidths, regulating data flow to a client, and supporting personalized caching to improve restart times. Network flow buffering is easily implemented as a Customizer in a middleware architecture we have developed (and reported on in WWW02) called the Web Stream Customizer Architecture (WSCA). The WSCA has a number of desirable features for supporting network flow buffering, including the dynamic deployment and relocation of software intermediaries between clients and servers that allow them to customize Web transactions, an extended usage model that supports customization of HTTP streaming and non-HTTP communication, and two points of control (one at the intermediary and one at or near the client) which can be used to supply feedback on conditions at the client to improve a network flow buffer's performance.

1. INTRODUCTION

A 1999 study of nearly 83 million HTTP accesses from thousands of clients at a major university in the U.S. put forth a lower bound of 18%-24% for the proportion of all traffic that was dedicated to multimedia [17], a number that has likely increased significantly in recent years as hardware has become more powerful and broadband Internet access has gained in popularity. A follow-up 2001 study of video traffic from thousands of clients at the same university found that 93% of video sessions were for videos of length 10 minutes or less, most being between 2.5 and 3.5 minutes [4]. As Web-based access to unicast streaming of pre-recorded video clips becomes increasingly popular, users are encountering a number of problems with the performance of video playback. In today's Internet, under ideal conditions (e.g., a powerful desktop client accessing video from a high bandwidth server over a high bandwidth, broadband connection without congestion on the route from client to server), smooth video streaming is already a reality. However, when these conditions are not met, the user will often experience extended startup delays and frequent interruptions in playback. Playback problems can originate anywhere along the path from client the server, due to inadequate client memory, network conditions such as wireless connection failures or congestion in the WAN, and servers failing or lacking adequate resources to maintain the required stream bandwidth.

There exist a number of specialized video streaming protocols. However, HTTP-based video streaming is popular and supported by most video players for a number of reasons. One reason is the use of HTTP proxies from behind firewalls that allow access to the outside world; another is the fact that it is easier for common Internet users to place video clips on Web sites provided by their ISP as they may not have direct access to a server, or they may lack the expertise to install and administer their own video server. In the former case, the video can be streamed directly over HTTP, or a streaming protocol can be tunneled through HTTP. However, in the latter case, the server dictates that HTTP be used directly. Specialized streaming protocols are ideal for dedicated video systems such as video conferencing and video-on-demand servers. In this paper, we present an approach to improving the performance of streaming video using direct HTTP transfers, making it a more viable alternative to specialized streaming protocols when viewing Web-based video clips, and in some cases making it a superior approach.

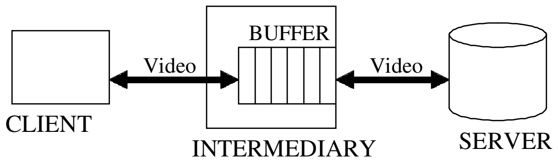


Figure 1. Network Flow Buffering.

To mitigate performance problems of HTTP streaming, we introduce the simple notion of *network flow buffering*, where a simple application-layer buffer is dynamically deployed between a Web client and server to actively or passively regulate the flow of streaming video data (see Figure 1). In addition, it may also cache data for future access. Network flow buffering can provide benefits such as reducing playback startup latency, reducing the frequency and duration of interruptions to video playback, and reducing the time window in which the application is susceptible to failures in the network or server. It can be used in a number of capacities such as supplementing the client's memory, smoothing bandwidth differences between the WAN and LAN connecting the client and server, regulating data flow to the client, and storing data beyond the duration of the connection for caching purposes. By regulating data flow to the client, network flow buffering can effectively convert a normal HTTP video session between the server and intermediary into a specialized streaming protocol between the intermediary and the client.

Normally, when an application-layer intermediary such as an HTTP proxy is being used, the operating system of the proxy host will buffer network data as it arrives on the network before it has been read by the application. One might conclude that buffering at the application layer is therefore redundant. However, the OS buffer is generally not adequate because typically, it will be too small to provide any benefit for large transfers, such as video clips of size 1 MB or more.

Our motivation for network flow buffering arose directly from practical experience with commercial video players, including Real™, Windows Media™, and Quicktime™, and the observation that these players often spend significant amounts of time buffering data both before playback begins and when playback is interrupted due to underflows. Network flow buffering is entirely transparent to the video player and the video server, allowing it to improve existing video applications as an alternative to using specialized streaming protocols or introducing new streaming protocols.

Note that our approach is orthogonal to related research on video smoothing (discussed further in the section on related work), such as approaches that optimize for known video encoding characteristics and hard network limitations, as we take a client-specific approach, focusing on adapting to changing network conditions under client constraints, while being independent of server behavior, and we focus only on HTTP streaming.

Although buffering is a common approach to improving the performance of distributed systems and its benefits are well known, we make the following unique contributions:

- We focus on streaming video clips, identifying and modeling a number of different scenarios in which buffering improves the playback of streaming video directly over HTTP, some of which require passive buffering and others which require the outward flow from the buffer to be regulated.

- We take into consideration the location of the buffer and consider the fact that the buffer may need to be dynamically relocated to support mobile users.

- We consider that the buffering code itself may be distributed into two points of control, which can be used to hone performance, or combine two usage scenarios to reap the benefits of both.

The Web Stream Customizer Architecture (WSCA) [13] has a number of desirable features that make it an ideal platform for implementing network flow buffering. It supports the dynamic deployment and relocation of software intermediaries between Web clients and servers, allowing them to customize Web transactions. It also has an extended usage model that supports customization of HTTP streaming and non-HTTP communication. The WSCA is transparent to clients and servers, and provides two points of control, one point of control at the intermediary, and one point of control running on or near the client. This client-based component can be used to supply feedback to a network flow buffer on conditions at the client to improve the application's performance.

In the remainder of the paper, we describe some of the most common and useful network flow buffering scenarios in Section 2. In Section 3, we present performance evaluations of two of these scenarios. In Section 4, we discuss the implementation of network flow buffering within the WSCA. We review related work in Section 5, and present conclusions in Section 6.

2. Network Flow Buffering Scenarios

The optimal location for network flow buffering depends on the network conditions. Ideally a network flow buffer (NFB) is located just beyond the "problem hop," a boundary point where there is a significant change in network performance characteristics. This location will often be at the LAN/WAN gateway, such as when the client has a low-bandwidth wireless connection (depicted in Figure 2). However, when the problem is in a WAN hop, there may not be a suitable location for hosting an NFB, so the LAN/WAN gateway may serve as a practical alternative.

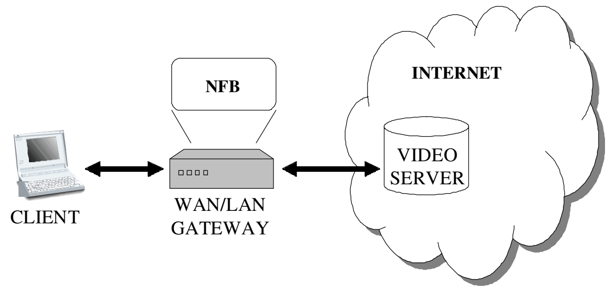


Figure 2. A Network Flow Buffer on the LAN/WAN Gateway.

The following five scenarios highlight the benefits of network flow buffering. The description of each scenario below is given in the context of the NFB being located at the LAN/WAN gateway, and is accompanied by a pair of charts depicting the data receipt and video playback with and without using network flow buffering.

2.1 Pre-buffering

By pre-buffering, we mean buffering that occurs before video playback begins. Typically a streaming player will buffer a few seconds of video, even when data is arriving at a high rate, so that it can handle possible future network or server problems that may cause data to arrive late. In addition, when the data arrives at the client at a rate that is lower than the video playback rate, the client needs to buffer more to ensure continuous playback once playback begins.

Pre-buffering is especially useful when the WAN is the bottleneck and the client has limited memory. In Figure 3, the video playback and network data rates are modeled as constants to simplify the example. The limited-memory client cannot buffer adequately to support smooth playback without interruption, even though the network problems originate in the WAN. However, an NFB near the client can supplement the client's buffer by buffering some extra video before it begins sending data to the client. The client only needs to buffer enough data to sustain the video as though it is originating at the NFB. Conditions amenable to pre-buffering may occur if there is congestion in the WAN, or if a high bit-rate video is being played over a high-speed LAN such that the WAN is the bottleneck.

Like the client, the NFB cannot predict future network conditions over the course of the entire video, but it may buffer based on current conditions, based on feedback from the client about its memory availability and effective bandwidth, based on user settings, or based on a heuristic such as using data from past streaming sessions to predict future performance. Ideally, the player would know about the buffer and they could cooperate so that the player does not interpret the pre-buffering at the buffer as a network problem. Instead, many players can be configured to wait for up to a few minutes before giving up on a server connection, so this would also allow for pre-buffering to occur even without player cooperation.

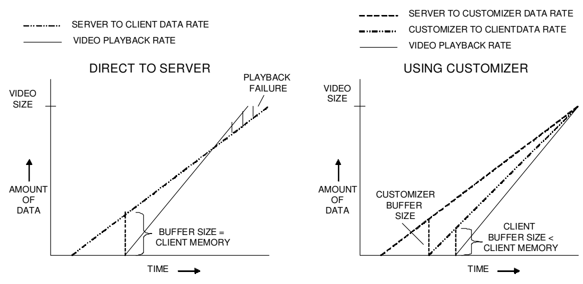


Figure 3. Pre-buffering.

2.2 Network-Differential Buffering

Network-differential buffering is a form of passive buffering that takes place during video playback, and is useful when the WAN bandwidth is generally higher than the LAN bandwidth. The goal of network-differential buffering is to take advantage of the WAN bandwidth to get the data from the server to the NFB as quickly as possible so that a WAN or server failure will not interrupt video playback. It is passive because the buffer fills up automatically due to the difference in bandwidths between the WAN and LAN. The NFB retrieves data from the server faster than data flows from the NFB to the Client. The net effect is that when the NFB is being used, data leaves the server sooner to be buffered at the NFB. Without the NFB, the data would remain on the server due to the lower effective bandwidth between client and server, forcing the server to send data more slowly.

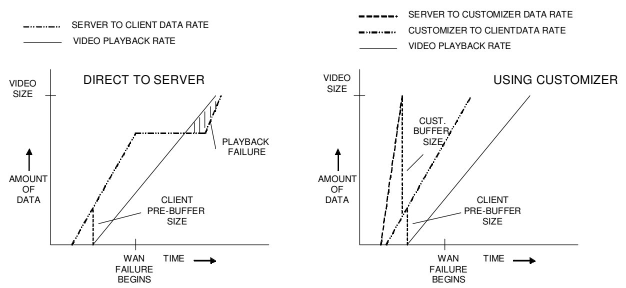


Figure 4. Network-Differential Buffering.

This scenario is depicted in Figure 4. Network-differential buffering may be the result of an actual difference in the WAN and LAN capacities at the time of the streaming, or the result of the inability of the application to take advantage of the full bandwidth. For example, the LAN bandwidth may be adequate, but the client has limited buffering memory available, forcing the data to be transferred from the server at a lower rate if no NFB is used. In Figure 4, when the client obtains the video stream directly from the server, the player detects that the bandwidth is adequate to sustain the video playback, so it only pre-buffers a moderate amount of data so as not to delay starting playback for too long. However, a server failure lasts long enough to cause buffer underflow at the client and prevent the client from maintaining normal video playback. When the NFB is used, the client pre-buffers the same amount. However, the NFB receives the video quickly enough that it has finished downloading the entire video before the server failure occurs. This allows it to mask the failure from the client. Because of the high WAN bandwidth, the NFB does not need to pre-buffer very much, so the initial delay before playback starts is not much greater than what it would be if the client were to stream directly from the server.

2.3 Regulated Split-Stream Buffering

Regulated split-stream buffering allows the client to reap the benefits of a specialized streaming protocol even when the video is on a Web server. However, it has the added benefit that it decreases the window of time in which the playback is susceptible to server or WAN failure, making it an improvement over specialized streaming protocols in the case where the server is not heavily loaded. Regulated split-stream buffering is similar to network-differential buffering except that the lower data rate from the NFB to the client is explicitly regulated by the NFB. In effect, the NFB acts as a regulated buffer, downloading data as quickly as possible from the server and regulating the flow of data to the client. This is typically implemented as a split stream, i.e., the Server-NFB protocol differs from the Client-NFB protocol. This is especially useful when conditions at the LAN and client are such that a streaming protocol would be highly beneficial (e.g., high error rate, low memory), but conditions at the WAN would support a fast download by the NFB. In this case, if the NFB can establish a direct HTTP connection with the server, the NFB can download the data at full speed while streaming the data at a regulated rate to the client. The buffer will fill up quickly, yielding the same benefits as network-differential buffering, as shown in Figure 4. Later, we will show the benefits of this scenario with some experimental results, comparing it to using a specialized streaming protocol along the entire path from client to server.

2.4 Smoothing

In the context of the NFB, smoothing refers to using the buffer to mask (from the client) dynamic changes in relative bandwidth between the WAN and LAN that cause the effective bandwidth to be lower than the video playback rate. When the bandwidth is bursty and, on average, lower than the video rate, the client player will have to occasionally interrupt the playback to refill its buffer, even if it has adequate memory since it will be unable to predict the optimal pre-buffering amount (it may also determine that the estimated optimal pre-buffering will take longer than the user is likely to be willing to wait, and instead choose to intentionally rebuffer during playback).

Smoothing works best when the WAN and LAN are highly variable relative to each other, i.e., sometimes the WAN bandwidth is significantly higher than the LAN bandwidth, and sometimes the opposite is true. The effective bandwidth from the server to the client will always be limited to the minimum of the WAN and LAN bandwidths. If the WAN bandwidth is higher than the LAN bandwidth, there is an opportunity for an NFB to buffer data, just as in the network-differential buffering scenario.

If at some future time the WAN bandwidth drops below the video playback rate and the LAN bandwidth increases, the NFB will be able to send the buffered data to the client, effectively hiding from the client the fact that the WAN bandwidth is not high enough to support the video playback rate. This can save the player from having to interrupt playback.

Smoothing is depicted in Figures 5 and 6. Figure 5 shows the various bandwidths during the video session. Each of the lines in the plot shows potential amount of data transferred over time, and hence the *slope* represents the bandwidth. Although the WAN and LAN have the same average bandwidth, the WAN bandwidth is bursty while the LAN bandwidth is smooth. The effective bandwidth seen by the client is also shown (calculated as the minimum slope of the WAN and LAN bandwidth plots at any time), and has a lower average. (The minimum slope is used because, again, the slope represents the bandwidth as a rate of data arrival over time.) In Figure 6, the video playback is depicted with and without an NFB. Without the NFB, the pre-buffering is inadequate. When the NFB is used, the same amount of pre-buffering at the client yields smooth video.

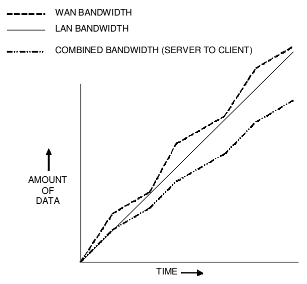


Figure 5. Bandwidths for Smoothing Scenario

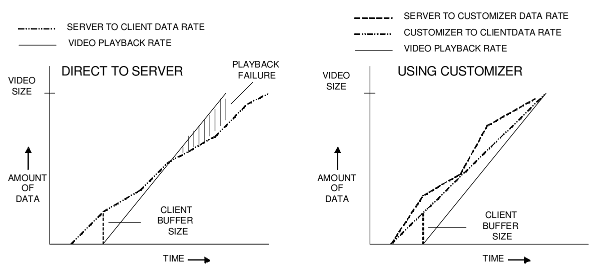


Figure 6. Smoothing

2.5 Personalized Caching

Network flow buffering can easily be extended to support caching. Caching helps low-memory clients by reducing the likelihood of interruptions during playback, and decreasing the time it takes a player to restart playback after the user fast-forwards, rewinds, or replays the video either by choice or as a result of a network outage. Retrieving the video from the cache reduces the latency, and since it is less likely that a network problem such as congestion occurs along the data path, it can also reduce the pre-buffering time at the client and the likelihood of the client having to interrupt playback to rebuffer.

Unlike traditional proxy caches, which aim to improve performance by caching videos that are popular among a number of clients, personalized caching is primarily aimed at reducing restart times for videos recently viewed by the user. The cache can be relatively small as, not only is it targeted for a single user, but if the desired goal is to reduce restart times resulting from interruptions to playback due to wireless connection failure, the cache only needs to store the most recently accessed video, and only until the client has successfully completed viewing the video.

An example of improving restart times with personalized caching is shown in Figure 7. The LAN bandwidth is higher than the WAN bandwidth. Without the NFB, after a LAN failure occurs, the user restarts the video, and must wait for the full server latency and for rebuffering based on the lower WAN bandwidth. With the NFB, the first time playback begins, the NFB buffers a little before sending to the client, and sends to the client at less than the maximum LAN bandwidth so that its buffer will not underflow. The extra latency added by the NFB reduces the amount of video that is displayed before the LAN failure. However, the NFB is able to receive the remainder of the video during the LAN outage. By the time the user restarts and fast-forwards, the entire video is cached on the NFB. The user sees only a small delay due to the NFB and to the small amount of rebuffering that occurs at the client. Since the entire video is buffered at the NFB, the full LAN bandwidth can be used between the NFB and client the second time through. In fact, other than compensating for jitter, buffering at the client is not even necessary; the client only buffers in anticipation of a potential failure.

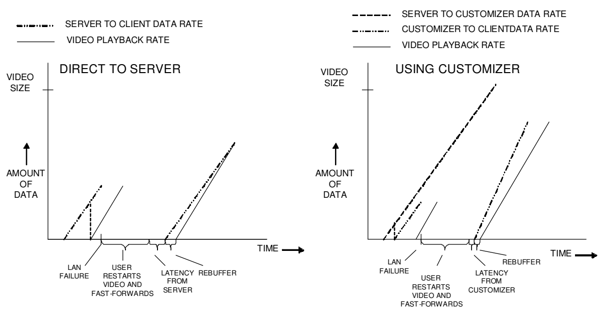


Figure 7. Personalized Caching

2.6 Combining Scenarios

In the introduction we note that the WSCA is an idea platform for implementing network flow buffering for a number of reasons, including the fact that it supports two points of control. This implementation is described in more detail in Section 4. Having two points of control supports an advanced NFB model in which two usage scenarios can be simultaneously active, one at a remote location beyond the "problem hop" and one closer to the client such as at the LAN/WAN gateway. For example, suppose an NFB is deployed just beyond a common problem area in the WAN near a server that is commonly accessed by the user. This is an ideal location for performing network differential buffering because the NFB is in good position to reliably and quickly receive the data from the server. However, suppose that the client is a low memory device and would benefit from pre-buffering at the NFB. In the case of pre-buffering, the NFB should be closer to the client so that there is minimal latency, since the buffer is essentially an extension of the client's memory. In this case, the ideal set-up is to have an NFB performing network-differential buffering at the remote location, and an NFB performing pre-buffering closer to the client such as at the WAN/LAN gateway.

3. EXPERIMENTS

We performed experiments to validate the benefits of smoothing and regulated split- stream buffering. The experimental setup for both experiments is shown in Figure 8. For the client, we used a notebook computer with a 500MHz Intel Pentium III processor running the RealOne™ Player on Windows 98. The server was a P3 933MHz PC running Windows 2000. Both an HTTP server and Real Helix Server™ were used. The intermediate "gateway" machine used for network flow buffering was a Pentium II 450MHz PC running FreeBSD.

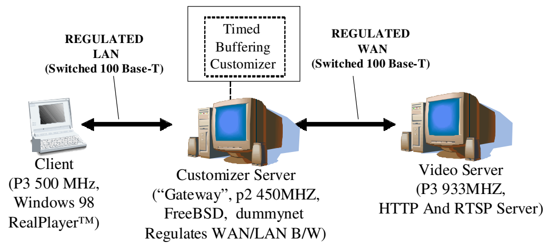

REGULATED LAN (Switched 100 Base-T)
Timed Buffering Customizer
REGULATED WAN (Switched 100 Base-T)
Client (P3 500 MHz, Windows 98 RealPlayer™)
Customizer Server ("Gateway", p2 450MHZ, FreeBSD, dummynet Regulates WAN/LAN B/W)
Video Server (P3 933MHZ, HTTP And RTSP Server)

Figure 8. Experimental Setup

To simulate a network with a given bandwidth, we used "dummynet" in FreeBSD (IP Firewall kernel module), which supports the creation of pipes to control bandwidth between two communication endpoints. We ran dummynet on the gateway machine, controlling bandwidth between itself, the client, and the server. We also ran tcpdump, a Unix program that logs IP packets and their sizes, including the amount of user data, on the gateway machine to track network communication. We also ran a Windows version of tcpdump called windump at the client, to precisely track the arrival of data at the client.

Measuring the video playback and buffering in the video player was a bit more difficult, as we did not have access to the internals of the Real Player™. Thus, we assumed a constant bitrate playback rate based on the attributes of the video being used, and we manually timed the buffering periods of the player based on information provided via its user interface, which indicates when it is buffering. So, for example, a general way to estimate the size of the player's buffer is to subtract the amount of arriving data reported by windump from the amount of video data that has been displayed. The amount of video data is calculated as the amount of time the video has played so far, which is displayed by the player, multiplied by the bitrate of the video reported by the player. We used the same video clip for all experiments: it was 3 minutes and 22 seconds in length, with a bitrate of 308Kbps (38.5 KB/s).

3.1 Experiment 1: Smoothing

In this experiment, we show how a NFB (network flow buffer) can provide smooth playback that would otherwise be interrupted multiple times. We also demonstrate that in addition to the performance improvement, using the NFB requires no more application-layer buffering than is required when streaming directly from server to client. In fact, in circumstances when smoothing works well, it is normally expected to use less.

We used dummynet to simulate highly bursty LAN and WAN bandwidths (over time). The bandwidths are shown in Figure 9. The LAN and WAN bandwidths repeat the same 24-second cycles, but start at a different point within the cycle. For each, the bandwidth changes every 2 seconds, half of the cycle is spent above the video rate and half of the cycle is spent below the video rate. The individual averages for the WAN and LAN for a cycle are 41.75 KB/s, which is above the average playback rate of 38.5 KB/s. However, the combined bandwidth, calculated as the lower of the two at any point in time, is 25 KB/s. As a result, without smoothing, we expect the video player to have to interrupt playback so that the network can catch up to the video. The benefits of smoothing are illustrated below.

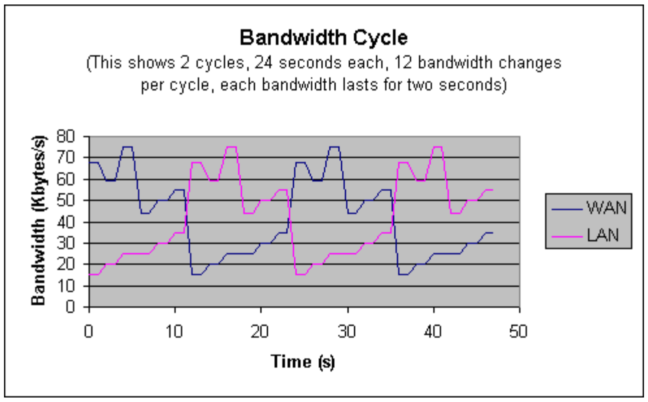


Figure 9. Bandwidth For Smoothing Experiment

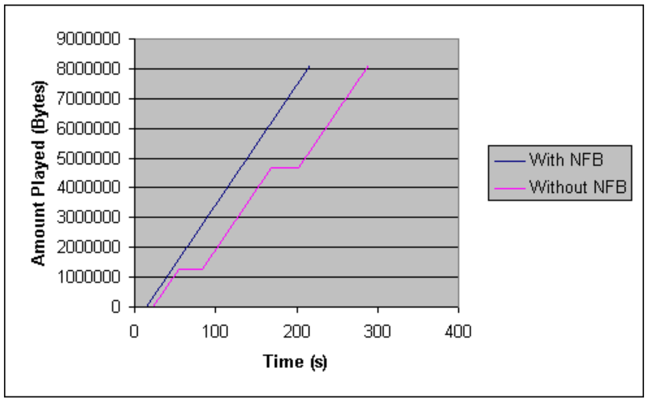


Figure 10. Playback With/Without Network Flow Buffering

We performed two video trials. For both trials, the NFB host was configured to be the HTTP proxy for the player, and the player was configured to stream directly over HTTP from a Web server. In one trial, the NFB application simply acted as a pass-through streaming proxy without buffering. Going through the proxy simulates the data passing through a network gateway between the LAN and WAN, and allows the proxy host to run dummynet and control the LAN and WAN bandwidths. In the second trial, the NFB smooths the data flow between the server and client. It is implemented very simply using two threads. One thread reads data from the connection to the server as it becomes available and writes it to a data buffer. Another thread reads data from the buffer and writes to the client. Both threads use synchronous network I/O. When the WAN bandwidth is higher than the LAN bandwidth, the buffer will increase in size. When the WAN bandwidth drops and the LAN bandwidth surpasses it, data will be available in the buffer to send to the client to compensate for the lack of data arriving from the server. Figure 10 shows the video playback as amount of data played over time with and without NFB.

When the NFB is being used, there are about 15 seconds of pre-buffering. Once playback begins, the video plays smoothly for its entirety without any additional buffering by the player. This is due to the smoothing effect of the buffer, which allows a higher average bandwidth to be sustained to the client. Without the buffer, the player pre-buffers for about 22 seconds. During playback, it interrupts the video twice to refill its buffer: one buffer refill begins 54 seconds from the time that the start button was pressed on the player and has a 30 second duration, and the second interruption begins a total of 168 seconds after the start button was pressed and has a 35 second duration. This sums to 65 seconds of buffering after playback has started and 87 seconds of total buffering including pre-buffering, as compared to just 15 seconds with the NFB. The extra buffering that occurs when there is no smoothing is required because the player's buffer is suffering from underflow due to the fact that the average bandwidth is lower than the video bit rate. Note that even if the player had perfect knowledge of the future, it would have had to delay the start of the video by about 87 seconds to ensure smooth playback. Not only would this frustrate the user, it would also require a buffer size of roughly 3MB (87s multiplied by 38.5 KB/s). Given that the player cannot anticipate the highly bursty nature of the network traffic, or that it may not be desirable to delay the start of the video for so long and force the user to wait, or to reduce memory consumption, the player is forced to interrupt the video for two periods of half a minute or more to complete the playback.

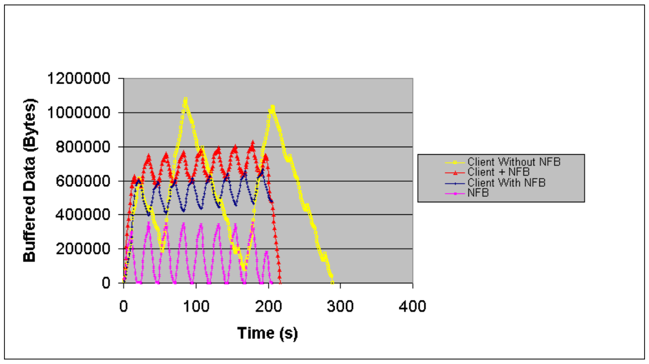


Figure 11. Buffering During the Smoothing Experiment

In Figure 11, we show the amount of buffering by the player in both scenarios, and we show the application-layer buffering at the NFB as well as the combined client and NFB buffering when the NFB is being used. The line labeled "Client Without NFB" represents the amount of data buffered by the player over time. As described above, this is calculated as the difference between the video bit rate and the incoming data rate, taking into account buffering periods when the video is not playing. The first spike represents the pre-buffering before playback begins. Since the video plays faster than the incoming data rate, the client's buffer steadily reduces in size until the player decides to refill the buffer by pausing video playback. This occurs twice, corresponding to the two remaining peaks. The maximum buffer size for the player is 1076886 bytes.

The line labeled "Client + NFB" is the sum of the buffering at the Client and the NFB when the NFB is being used. This is further broken down into its two components, the "NFB" line represents the buffering by the NFB, and the "Client with NFB" line represents the buffering at the client when the NFB is being used. The bursty nature of these three lines is a direct result of the bursty bandwidth. The NFB buffer will increase in size when the WAN bandwidth is greater than the LAN bandwidth, and decrease in size when the LAN bandwidth is greater than the WAN bandwidth. This happens approximately every 12 seconds due to the bandwidth schedule depicted in Figure 9. The "Client With NFB" line has a slight upward trend because, when smoothing is active, the average bandwidth over the course of the video is higher than the video bit rate. Hence, despite the burstiness, on average, data is arriving at the client slightly faster than it is being displayed on the screen. The client buffer in this case is also bursty due to the burstiness of the bandwidth. When the LAN bandwidth is lower than the video rate, its buffer will reduce in size, and when the opposite is true its buffer will grow. The maximum combined buffering (client + NFB) peaks at 826419 bytes, less than the maximum buffer size at the client when the NFB is not used. This is due to the fact that without the NFB the effective bandwidth is lower, so more buffering is required to compensate.

3.2 Experiment 2: Regulated Split-Stream Buffering

In this experiment, we compare regulated split-stream buffering to the use of a specialized streaming protocol, and demonstrate how regulated split-stream buffering can perform better because it can mask WAN and server failures from the client. The client, NFB host, and server hosts are the same as in the prior experiment. The WAN bandwidth is set at 900 Kbits/s and the LAN bandwidth at 450 Kbits/s. After 20 seconds, the WAN bandwidth drops to 100 Kbits/s for 25 seconds, then goes back up to 900 Kbits/s. This simulates a lapse in WAN or server performance. There were two video trials, representing streaming without the NFB and split-streaming with the NFB. The same video is used as in the prior experiment, which has a bitrate of 308 Kbits/s. The first trial shows the behavior without the NFB. The client streams the video from the server using RTSP. However, to simulate the network gateway and regulate the WAN and LAN bandwidths, the client uses a streaming RTSP proxy running on the intermediate FreeBSD machine. All data passes through the proxy but the proxy does not do any application-layer buffering; it simply forwards data to and from the client. We used a reference implementation of an RTSP proxy available from http://www.rtsp.org/2001/proxy/.

The second trial simulates the behavior of a regulated buffer. To mimic a NFB that acts as an RTSP server to the client, we split the functionality into separate components and used the hard disk as the buffer. We installed a full-blown Real™ server on the FreeBSD machine. We first start a direct HTTP retrieval of the video from the server to the disk on the FreeBSD machine. We then immediately push play on the client video player, which causes the client to start streaming the video from an RTSP server also running on the FreeBSD machine. The RTSP machine is configured to read the file from the location where the HTTP transfer is saving the file. The net effect is that the file is downloaded at maximum speed from the server to the NFB Server, and streamed at approximately the video playback rate from the NFB Server to the client.

Figures 12 and 13 show the results of the trials with and without the NFB, respectively. In each figure, three lines are shown. The lines labeled "Proxy: Incoming Data" and "Client: Incoming

Data" show the arrival of data over time at the proxy and client respectively. When the NFB is being used, the "Proxy: Incoming Data" line is the rate that the HTTP data arrives at the FreeBSD machine. When the NFB is not being used, this line represents the arrival of data at the RTSP proxy on the FreeBSD machine. The line labeled "Video Playback" shows the amount of video data displayed by the video player over time in both scenarios. It should be noted that the proxy receives more total data than the client. This is most likely the result of the streaming protocol selecting what data to send to the client, unlike direct HTTP where all of the data is automatically downloaded. The video may have been encoded with multiple quality levels so all of the data is not needed.

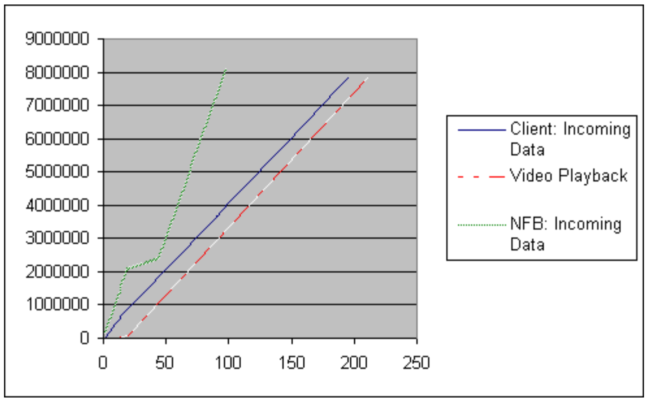


Figure 12. Playback with Regulated Split-Stream NFB

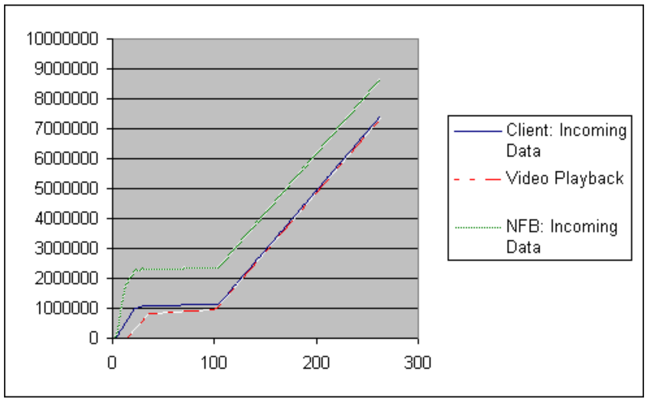


Figure 13. Playback without Regulated Split-Stream NFB

When the NFB is used, playback is smooth. The proxy retrieves the data quickly enough so that when the WAN lapse begins, there is enough data already cached at the proxy to maintain the full video rate streaming to the client. However, without the NFB, the WAN lapse causes a long interruption in video playback because the reduction in the incoming data rate allows the video playback to catch up to the network. Notice that the streaming protocol does not deal with the sudden drop in bandwidth very gracefully. It actually slows down the streaming below 100 Kbits/s; in fact it averages below 8 Kbits/s for roughly 75s, despite the lapse only lasting just 25 seconds. Also, the proxy receives much more data than the client. In fact, this gap is significantly larger than the corresponding difference when the NFB is used. This is most likely due to the streaming protocol reducing the quality of the video during the WAN lapse by requesting that not all of the video data be sent.

4. CUSTOMIZER IMPLEMENTATION

In this section, we describe the Web Stream Customizer Architecture, a framework for implementing network flow buffering based on past work we reported in [13]. This is certainly not the only possible framework for implementation, but it is instructive as a guide to requirements and implementation strategy. Specifically, the WSCA is an ideal platform for implementing network flow buffering because it meets the following requirements:

- **Transparency:** To promote ease of deployment, the use of network flow buffering does not require any changes to existing clients and servers. The WSCA supports transparent deployment and execution of intermediaries between the client and server using a proxy-based approach.

- **Flexibility of Location and Dynamic Deployment:** The best location for network flow buffering depends on a number of factors, such as where the likely problem areas in the network may occur and the location of the user. In the case of mobile devices, the location of the user is not known in advance. It also may be useful to relocate the buffer as the user moves. In the WSCA, Customizers are deployed dynamically and can be dynamically relocated.

- **Two Points of Control:** Network flow buffering benefits from knowledge of conditions at the client, such as available memory and bandwidth. Knowledge of the client's LAN bandwidth can be a factor in determining how much data to buffer at the intermediary. Client feedback is supported by the WSCA, while maintaining transparency by providing a Customizer with two points of distributed control. Having two points of control also allows for combining usage scenarios as described in Section 2.6.

4.1 Architecture Overview

The WSCA expands upon the notion of fixed Web proxies by using a limited form of mobile code to allow dynamic deployment of customization modules called *Customizers* to a host server called a Customizer Server (CS). Figure 14 presents an overview of the WSCA, with two Customizers running on a single CS. The node labeled CIS is the Client Integration Server. This provides a service running on or near the client that allows client Web requests to be forwarded to a Customizer using the HTTP proxy mechanism. However, Customizers are not limited only to handling of HTTP transfers. A Customizer with network privileges can use the initial web request that starts a non-HTTP communication to set itself up as a proxy for the requested service.

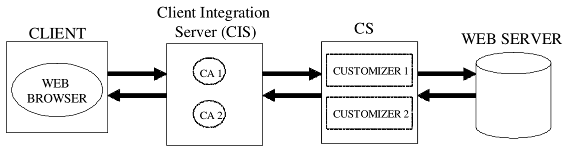


Figure 14. The WSCA.

One of the important features of the WSCA is that it supports two points of control. For each active Customizer, the CIS hosts a helper module called a Customizer Assistant (CA). The CA cooperates with a Customizer by providing functionality that must be at or near the client, such as for compression/decompression, encryption/decryption, and feedback to the Customizer on conditions at the client, such as available memory and bandwidth.

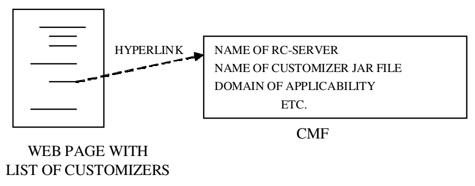
Network flow buffering can be implemented by having the Customizer do the buffering at a dynamically chosen location, while the CA provides bandwidth and memory feedback to help the Customizer make buffering decisions.

4.2 Customizer Deployment

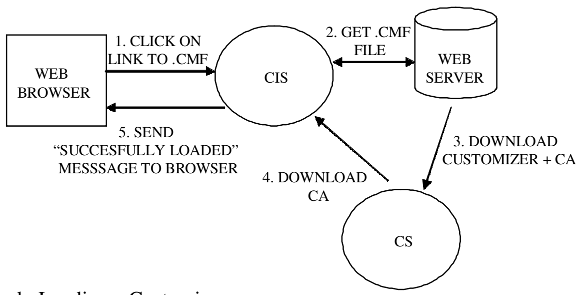
The CIS integrates the Web browser with the WSCA by acting as an HTTP proxy that can intercept all of the browser's requests. Once the browser is configured to use a CIS as its proxy, the user can *load* Customizers by clicking on special hyperlinks in Web pages. A Customizer is first made available by placing it on a Web server, and creating a separate Customizer Meta File (CMF), which is also placed on a Web server.

When the hyperlink is clicked, the Customizer is seamlessly loaded by the CIS. The CIS gets all the information it needs to load the Customizer from the CMF. This information includes:

- The hostname of the machine running a CS that will run the Customizer
- The name and location of the Customizer and CA.
- Initial configuration parameters for the CA and Customizer
- The *Domain of Applicability*, which specifies the sites (part of the URL) to be acted upon by the Customizer
- Optionally, a URL for the Customizer's configuration page.



a. A Web Page of Customizers With Links to CMFs



b. Loading a Customizer

Figure 15. Customizer Loading.

Figure 15a shows a web page with links to CMFs. Figure 15b shows the process of loading a Customizer by clicking on a hyperlink to a CMF. When the link is clicked, the CIS intercepts the request and retrieves the CMF from the Web server. Once the CIS has received the CMF, it can contact the CS, which will download the Customizer and CA, and then send the CA to the CIS. The CIS then sends a message to the browser to inform the user that the Customizer was loaded.

Note that in this example, the CMF and Customizer are collocated, but this is not required.

4.3 Customizing Streams

Once a Customizer is loaded, the CIS will forward HTTP requests, from the browser to the Customizer, that fall within the Domain of Applicability specified in the CMF used to load the Customizer. The basic Customizer usage model allows Customizers to act on HTTP requests. In the *basic model*, the CS handles all HTTP communication and uses a callback API to pass buffers to the Customizer, which it can modify. This model uses soft state, in that the CS is not required to maintain session state, or even actual Customizers, in between separate HTTP transactions. However, this model does not allow direct streaming from the Customizer. For applications such as network flow buffering, there is the *extended model*, which allows Customizers to handle arbitrary network communication that is initiated from within a Web page, such as video streaming. To enable this, Customizers that are trusted by a CS are given privileged access to resources that are persistent across multiple HTTP transactions, including memory for data, hard disk storage, and threads of execution. The Customizer must also have network I/O privileges.

For example, for an NFB Customizer to act on a video stream, either the video player can be configured to use the CIS as its HTTP proxy, so that its HTTP requests for videos will go through the Customizer, or the Customizer can be used to intercept the request before the video player is loaded as follows. When a Web server replies to a request with a page linking to a stream or a request for a metafile specifying the location of a multimedia file, the Customizer modifies the location of the video stream. It replaces the identity of the source of the video with itself, so that the client's video application tries to retrieve the video stream from the Customizer. The Customizer then acts as a proxy for the video streaming. The Customizer appears to the video server to be the video client, while the client sees the Customizer as the video server. This allows the Customizer to buffer the video data.

4.4 Dynamic Relocation

Since being presented at WWW02, the WSCA has been expanded to support dynamic Customizer relocation [14]. Due to the soft-state model of Customizer execution, Customizer relocation can be achieved by dynamically reloading the Customizer at a new CS. Customizers using the extended execution model may require the user to re-initiate the most recent session after the relocation. To support dynamic relocation, there is a special type of CS, called the Personal Customizer Management Server (PCMS), which can be run on a user-owned host or a host on which the user has an account (such as a desktop at work or at home). The PCMS serves as a central repository for the user's Customizers and can act as a control point for managing Customizer deployment and dynamic reloading of a Customizer to a new location. For example, as the user of a mobile device moves, the NFB could follow the user to remain near the current wireless gateway being used by the device.

5. RELATED WORK

There is a large body of research on smoothing compressed video streams by sending frames to the client buffer ahead of their playback time to reduce the burstiness of bandwidth requirements. In this approach, a video transmission schedule is calculated at the server based on the bit rate over the course of the video, the server bandwidth, and the amount client memory available for buffering. Here we discuss two notable contributions. In [10], the smoothing takes into account the possibility of having multiple intermediary buffers (such as proxies) along the path between server and client, and considers bandwidth limitations of network hops among the intermediary buffers, the client, and the server. In [5], the constraint of minimizing peak server bandwidth is relaxed to keep the client and server in closer synchronization, which better supports interactive functionality such as fast-forward and rewind.

The above techniques are based on static measurements; they take into account known video, network, and host characteristics and calculate a bandwidth schedule. Optimal smoothing can be approximated for live video by calculating the transmission schedule over a window of time based on the currently available video data in conjunction with predictive techniques [12]. However, we are attempting to solve a different problem, of dynamic changes in network performance and availability as a result of such factors as wireless network variability and congestion, for HTTP streaming. Our approach is transparent to the server so it does not require existing servers to change.

Another area of video research involves using intermediaries, either fixed or dynamically deployable, to improve video streaming using techniques such as transcoding and protocol adaptation for unicast or multicast [2, 3, 6, 7, 8, 9]. These approaches differ from ours in that network flow buffering does not do any transcoding of the video data. This type of functionality is more CPU intensive than buffering, which is I/O bound. Certainly, CPU intensive approaches such as transcoding can be used in conjunction with network flow buffering. If the CPU is powerful enough, and enough CPU cycles are available to the NFB, data can be transcoded as it is removed from the buffer before being sent to the client.

Another approach to improving video performance is to use proxy caching, where proxies cache all or part of commonly accessed videos to improve performance for a group of clients. A number of approaches perform partial caching of video streams to improve performance, while reducing resource consumption at the proxy [11, 16, 18]. Other approaches use cooperative caches to improve hit rates as well as reduce server load and WAN bandwidth utilization [1, 15]

Whereas proxy caching benefits a client that requests a video that is popular enough among multiple clients to remain in the cache, our approach is more client-specific in that it is deployed on behalf of a particular client and does not depend upon cache hits to be effective. Furthermore, our approach can be used in conjunction with proxy caching to improve streaming performance between the proxy and the client in cases where the proxy is not at the LAN/WAN gateway.

6. CONCLUSIONS

We have presented network flow buffering, a simple technique that can significantly improve the performance of direct HTTP streaming of pre-recorded web-based video clips. Network flow buffering can smoothen differences in WAN and LAN bandwidth, supplement the client's buffer, regulate data flow to the client while masking WAN and server failures, and support personalized caching to reduce latency and restart times. To implement network flow buffering, the Web Stream Customizer Architecture is especially useful, as it supports dynamic deployment and relocation, as well as two points of control.

Many network flow buffering usage scenarios aim to mask server and WAN failures from the client by obtaining data from the server as quickly as possible. Traditionally, this was not a practical approach due to the cost and limited availability of client memory. Today's desktops have enough combined memory and disk space to efficiently buffer large video streams, and in such cases the client capabilities may not be a consideration for what type of protocol to use for streaming. Our approach provides additional buffering near the client for those client devices that lack adequate memory to support an HTTP-based streaming model. With network flow buffering, client constraints do not force a server to regulate bandwidth when it is not heavily loaded, even when the client is a low-memory device. This makes direct HTTP streaming a viable alternative to specialized streaming protocols, and in some cases, such as regulated split-stream buffering when the server is not heavily loaded, it is superior to using these protocols.

REFERENCES

[1] S. Acharya and B. Smith, " MiddleMan: A Video Caching Proxy Server," *In Proceedings of 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, NC, June 2000.

[2] E. Amir, S. McCanne, and R. Katz, "An Active Service framework and its application to real-time multimedia transcoding," *In Proceedings of ACM SIGCOMM*, pp. 178-189, Vancouver, Canada, August 1998.

[3] F. Baschieri, P. Bellavista, and A. Corradi, " Mobile Agents for QoS Tailoring, Control and Adaptation over the Internet: the ubiQoS Video on Demand Service," *In Proceedings of Symposium on Applications and the Internet (SAINT 2002)*, Nara City, Japan, January/February 2002.

[4] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and Analysis of a Streaming-Media Workload," *In Proceedings of of the 3rd USENIX Symp. on Internet Technologies and Systems (USITS)*, San Francisco, CA, March 2001.

[5] W. Feng, "Rate-Constrained Bandwidth Smoothing for the Delivery of Stored Video," *In Proceedings of IS&T/SPIE Multimedia Networking and Computing*, pp. 316-327, San Jose, CA, February 1997.

[6] Z.Lei and N.D.Georganas, "Rate Adaptation Transcoding For Video Streaming Over Wireless Channels"," *In Proceedings of IEEE ICME 2003*, Baltimore, MD, June 2003.

[7] J. Meggers, T. Strang, and A.S. Park, "A Video Gateway to Support Video Streaming to Mobile Clients," *In Proceedings of ACTS Mobile Communication Summit*, Aalborg, Denmark, October 1997.

[8] R. Mohan, J. Smith, and C.-S. Li, " Adapting Multimedia Internet Content For Universal Access," *In IEEE Transactions on Multimedia*, pp. 104-114, March 1999.

[9] W. T. Ooi, and R. van Renesse, "Distributing Media Transformation Over Multiple Media Gateways," *In Proceedings of ACM Multimedia 2001*, Ottowa, Ontario, Canada, October 2001.

[10] J. Rexford, and D. Towsley, "Smoothing variable-bit-rate video in an internetwork," *In IEEE/ACM Transactions on Networking*, pp. 202-215, April 1999.

[11] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," *In Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[12] S. Sen, J. Rexford, J. Dey, J. Kurose, and D. Towsley, "Online smoothing of variable-bit-rate streaming video," *In IEEE Trans. on Multimedia*, pp. 37-48, March 2000.

[13] J. Steinberg and J. Pasquale, "A Web Middleware Architecture for Dynamic Customization of Content for Wireless Clients," *Proceedings of the 11th International World Wide Web Conference*. Honolulu, Hawaii, USA, May 7-11, 2002.

[14] J. Steinberg, "The Web Stream Customizer Architecture: Improving Performance, Reliability and Security for Wireless Web Access," *Ph.D. Dissertation University of California San Diego Department of Computer Science and Engineerint.* In preparation.

[15] D. A. Tran, K. A. Hua, and S. Sheu, "A New Caching Architecture for Efficient Video Services on the Internet," *In Proceedings of IEEE Symposium on Applications and the Internet (SAINT 2003)*, Orlando, FL, USA, January 2003.

[16] Y. Wang, Z. Zhang, D. H.C. Du, and D. Su, "A Network Conscious Approach to End-to-End Video Delivery over Wide Area Networks Using Proxy Servers," *In Proceedings of IEEE INFOCOM'98*, San Francisco, CA, April 1998.

[17] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy., "Organization-Based Analysis of Web-Object Sharing and Caching," *In Proceedings of the 2nd USENIX Conference on Internet Technologies and Systems* (USITS), Boulder, CO, October 1999.

[18] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," *In Proceedings of the 10th International WWW Conference*, Hong Kong, May 2001.