Efficient Video Streaming using TCP

Puneet Mehra
EE228A Fall 2002 Final Project
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
pmehra@eecs.berkeley.edu

Abstract

While most of the traffic on the Internet today consists of TCP flows, conventional wisdom holds that TCP is unsuitable for video streaming applications due to its insistence on reliability and lack of throughput guarantees. However, most streaming protocols need to implement many features of TCP including congestion control and recovery from packet loss. Additionally, using TCP for streaming is often unavoidable when the target clients are located behind firewalls. This work seeks to dispel the notion that TCP is inherently unsuitable for streaming. We focus on the common case of streaming video to receivers whose *last mile* connections to the Internet are bandwith-limited and act as network bottlenecks. Users generally run multiple concurrent networking applications that compete for the scarce bandwidth resource. Standard TCP shares bottleneck link capacity according to connection round-trip time (RTT), and therefore may not provide streaming applications with the necessary bit-rate. In our prior work, we outlined a receiver-based bandwidth sharing system (BWSS) for allocating the capacity of last mile bottlenecks among TCP flows according to a user's preferences. This system does not require modifications to the TCP protocol, network infrastructure or sending hosts, making it easy to deploy. By breaking TCP fairness between flows on the access link, the BWSS can limit the throughput fluctuations of high-priority applications. In the present paper we utilize the BWSS to perform video streaming over TCP. We establish the benefits of our proposed approach over standard TCP through Internet experiments involving a prototype for the Linux operating system. Furthermore, we demonstrate scenarios in which a client using our bandwidth control system may actually obtain *better* performance than TCP-friendly UDP streaming protocols.

1 Introduction

The recent proliferation of broadband Internet access has fueled the increasing popularity of video streaming applications. From news clips on sites such as CNN.com, to video on demand from MovieFlix [1], the current Internet offers a much richer multimedia experience than in the past. While the majority of traffic on the Internet today is comprised of TCP [2] flows, conventional wisdom holds that TCP is unsuitable for "real-time" traffic due to its lack of throughput guarantees and insistence on reliability. The only alternative protocol to TCP is UDP, which does not retransmit lost data packets, or provide the congestion control [3] features offered in TCP. It is widely accepted in the networking community that congestion control is essential to the continued growth and stability of the Internet [4]. Consequently, much work has gone into the creation of streaming protocols built on top of UDP, which perform some form of congestion control and ensure fairness with the ubiquitous, competing TCP traffic [5]. A wide body of literature, including [6] and [7], attest to the technical challenges involved in creating such TCP-friendly rate control protocols. Furthermore, in many situations streaming over TCP is unavoidable, such as when client machines are located behind network firewalls permitting only inbound HTTP traffic.

This works seeks to dispel the notion that TCP is always unsuitable for multimedia streaming. We note the benefits of streaming with TCP, including its retransmission capabilities, congestion control, as well as natural fairness with other TCP traffic. Many streaming applications involve a certain amount of pre-buffering which allows them to absorb any jitter, or variation in packet delay, in data transmission. Hence retransmission delay and temporary bandwidth fluctuations due to packet loss can be accommodated through appropriate pre-buffering of the multimedia stream.

Our work focuses on the common practice of streaming video to receivers whose *last mile* connections to the Internet are bandwidth limited. Additionally, most users run multiple concurrent networking applications that compete for

the scarce bandwidth resource. In many cases, the limited bandwidth and operation of multiple networking applications cause the user's access link to act as a network bottleneck. Standard TCP shares the capacity of a bottleneck link among different flows according to their round-trip time (RTT), and therefore may not provide streaming applications with the necessary bit-rate they need to function properly. Hence we seek to address the shortcomings of TCP for adequate streaming to users running multiple TCP applications behind bandwidth-limited access links.

In prior work [8], we outline a receiver-driven bandwidth sharing system (BWSS), which allows a TCP receiver to control the manner in which the bandwidth of the user's access link is allocated to different applications, by adjusting the flow-control window advertised to the sender. This allocation is done based on pre-specified user preferences. In essence, the BWSS allows a user to break fairness among her own flows, and to partition bandwidth in an application-specific manner. The BWSS can be used to eliminate throughput fluctuations in TCP, which are detrimental for streaming applications. Furthermore, this system requires modest changes to a TCP receiver and does not require any modifications to the network infrastructure or to TCP senders, facilitating easy deployment. It is our claim that by using the BWSS to control the throughput of a streaming application using TCP, we can achieve efficient video streaming without resorting to UDP. This can be quite beneficial for the situations in which streaming applications are forced to use TCP to accomodate user firewalls. Furthermore, we contend that if there is congestion which is restricted to the user's access link, then by breaking the fairness among a user's TCP connections, it is possible to provide higher throughput for streaming applications than by using UDP streaming with a TCP-friendly protocol. This, in turn, leads to an overall "better" multimedia streaming experience for the user.

The BWSS can provide benefits in several scenarios for video streaming to bandwidth-limited receivers running multiple TCP applications. The BWSS effectively deals with the case when the overall access link bandwidth is reduced during the streaming session, and ensures that the streaming application receives its minimum required bit-rate. This bandwidth reduction may result from a user starting additional TCP connections during the streaming session. The BWSS state is reset whenever a flow is started or stopped, and hence the bandwidth allocated to the new flow is limited to prevent an adverse effect on the streaming connection. The BWSS is also effective if it is unable to control the traffic causing the congestion on the access link. For example, a user may start a UDP session on the host running the BWSS, or may initiate traffic to another host on a LAN connected to the Internet through the shared access link. When such congestion occurs, the BWSS responds by re-allocating the link bandwidth to ensure that the streaming application still has its minimum required bit-rate. There are two instances in which the BWSS fails to provide any benefits for video streaming. The BWSS is not useful if there is congestion on the path to the video source which limits the throughput of the streaming. Furthermore, the BWSS does not provide any benefits if there is congestion on the access link which reduces the available capacity below the required streaming rate. These cases require a reduction in the streaming rate and are an application-specific detail.

The rest of this paper is organized as follows. In Section 2 we discuss relevant prior work. In Section 3 we offer an overview of our receiver-based BWSS. In Section 4 we present results from different streaming experiments on the Internet. Finally we provide directions for future work and conclude this paper in Section 5.

2 Related Work

There have been several recent proposals which challenge the long-held belief that TCP is unsuitable for streaming applications. The authors of [9] provide a qualitative argument for the possibility of multimedia streaming using TCP. They note that client-side buffering can handle both the retransmission delays, and congestion control induced throughput variations, of TCP. Another technique proposed for streaming is Receiver-based Delay Control (RDC) [10], in which receivers delay TCP ACK packets based on router feedback. The authors of [10] attempt to mimic a constant bit rate (CBR) connection using RDC and also propose a layered streaming method. Our approach leverages our previously proposed BWSS described in [8] to provide a nearly CBR connection for the video stream, without any changes to routers or sending hosts. Time-lined TCP [11] is a proposal to support streaming over TCP by assigning deadlines to data passed to the TCP/IP stack in the operating system, and skipping any data which is past its deadline. Similarly, TCP-RTM [12] involves modifications to both the TCP sender and receiver which allow "stepping over" missed packets, thus avoiding the negative impact of TCP retransmissions. Our approach requires no modifications to the TCP protocol, or to senders. Futhermore, our method may be combined with approaches such as Timelined TCP or TCP-RTM to provide guaranteed throughput for a TCP connection, in addition to the real-time performance enhancements of these protocols.

3 System Description

We now provide an overview of our receiver-controlled BWSS for TCP applications, originally proposed in [8]. The goal of the BWSS is to allow a user to prioritize among applications by partitioning the limited access link bandwidth according to her preferences. The essential idea behind the system is to constrain the throughput of certain low-priority applications in order to provide additional bandwidth, if possible, for higher-priority flows as specified by the user's profile. Since our primary focus in this work is not the BWSS itself, but instead the utilization of the BWSS for efficient video streaming, our description of the system will be intentionally brief. We first discuss the bandwidth allocation model supported by the BWSS. We then briefly discuss the various components of our receiver-based solution. We conclude this section with a discussion of implementation issues related to our system prototype, used in conducting experiments, and highlight changes made to the system as described in [8].

3.1 Bandwidth Allocation Model

The bandwidth allocation model used by the BWSS recognizes two categories of applications: those that require a minimum throughput guarantee, and those which do not have any such requirements. For example, streaming applications are only viable above the streaming rate, and for a given pre-coded content at a fixed bit-rate, the perceptual quality does not significantly improve with throughput increases. Meanwhile, a file transfer application does not have any strict minimal bandwidth requirement, but may benefit from additional bandwidth. In order to capture the essence of such application preferences, we assign a priority, a minimal rate, and a weight to each TCP connection destined to the receiver. The bandwidth is allocated to applications according to the following algorithm. First, the minimal rate is provided to every connection, in decreasing order of application priority. Then, the remaining bandwidth is shared proportionally to the weight of the connection. This formulation captures the possibility that a receiver might prefer to starve low priority connections in order to improve higher priority ones. This is certainly desirable when sharing the bandwidth among all the connections makes it impossible to run any application well. It also captures the idea of weighted fair sharing of bandwidth between viable applications.

3.2 System Components

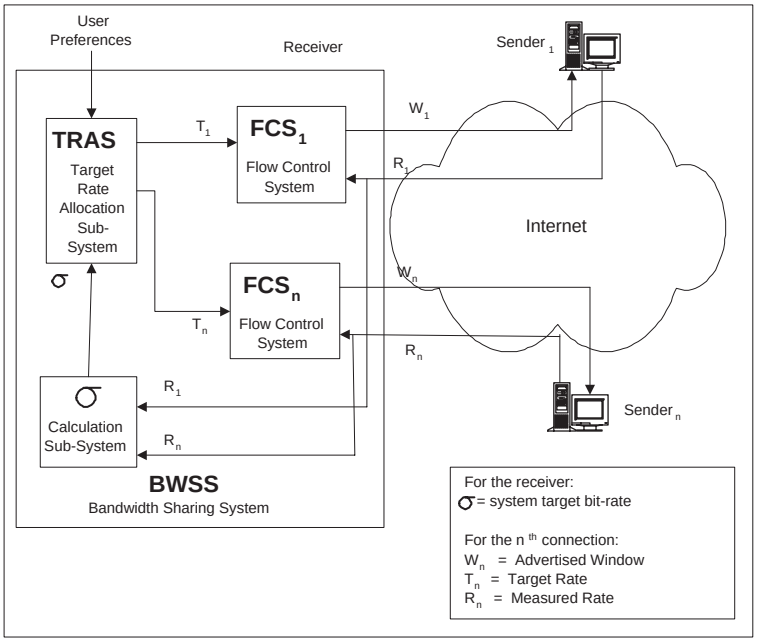

Figure 1: Receiver-based system for bandwidth sharing

We now briefly describe the different components of the BWSS, shown in the block diagram in Figure 1. More details of the BWSS are included in [8]. The main building block of the BWSS is the TCP Flow Control System (FCS),

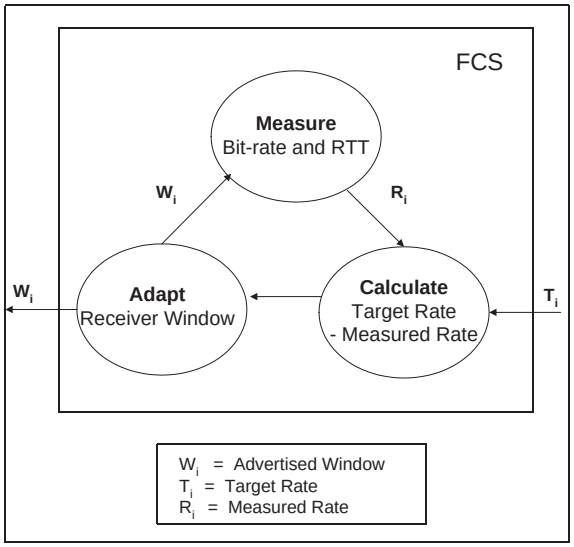

Figure 2: TCP Flow Control System (FCS)

which can constrain the rate of a given TCP connection to a particular bit-rate achievable given the flow's congestion window. As shown in Figure 2, the FCS is an iterative three stage process which consists of measuring the actual bit-rate of a flow, calculating the difference between the measured and target bit-rates, and then adapting the receiver's advertised window to achieve the desired target rate. A key component of the BWSS is the parameter σ, which is the system target bit-rate. This value represents the sum of target bit-rates allocated to different flows. As shown in Figure 1, given σ and external user-preferences, the Target Rate Allocation Sub-system determines the target bit-rates for each FCS in the BWSS. Since σ is responsible for the FCS target bit-rates, it is indirectly responsible for the actual throughput of these flows, and consequently it determines the overall link utilization. Hence the σ Calculation Sub-system is responsible for determining the optimal value of σ which achieves full link utilization. It converges to this value by increasing or decreasing the value of σ and considering the actual measured throughput of the different connections.

3.3 Prototype Implementation

We utilize a prototype of the BWSS for the Linux operating system to conduct various experiments involving video streaming. The BWSS is implemented as a shared library which overrides the *connect()* and *read()* functions of the C standard library, glibc, to provide the desired functionality of the BWSS. Additional details about the implementation of the BWSS are available in [8]. We will now highlight some changes made to the BWSS as described in our prior work.

In [8], the σ Calculation Sub-system made incremental changes to σ and then measured the impact of these changes on actual flow throughput before considering additional changes. The process of increasing or decreasing σ and then measuring the impact of these changes generally resulted in a fairly long convergence time, as shown in the NS-2 simulation results in [8]. Our simulations had indicated that the system needed to constantly "probe" the network for any possible increases in bandwidth, much like TCP, in order to ensure full utilization of the access link.

Subsequent operational experience with the BWSS over the actual Internet has led us to a much simpler formulation of the σ Calculation Sub-system, which is able to adapt to changes in flow throughput more rapidly. Specifically, the current σ Calculation Sub-System is simply reactive: it uses repeated measurements of the aggregate flow throughput to determine the value of σ. When the system is started, the initial aggregate TCP throughput is used as the initial value of σ. There are two different forms of congestion that the system must respond to: congestion which only affects a particular flow, and congestion which affects the entire access link. Furthermore, the system must be able to distinguish these two cases, and respond appropriately when congestion subsides. We will now examine the BWSS response to these two cases in more detail.

If there is congestion which affects a particular flow, the BWSS measures a throughput reduction for this flow, and

responds by allocating additional unused bandwidth from this flow to other connections. Specifically, if the measured throughput for a flow is below a fraction, α, of its target bit-rate, T, then σ is increased by $(1-\alpha)\cdot T$. The BWSS keeps track of a flow that experiences congestion, and when its measured throughput exceeds $\alpha \cdot T$, signalling the end of congestion, σ is set to the new measured aggregate throughput. To distinguish congestion which affects a particular connection from that which affects the entire access link, the BWSS uses a simple heuristic. If the throughput of at least half of the connections have been reduced below $\alpha \cdot T$, where T represents the target rate of a given flow, then the BWSS concludes that the congestion affects the entire access link, and responds by reducing σ to the measured aggregate throughput. After congestion subsides, the BWSS is able to measure an increase in the aggregate throughput. This increase in the aggregate throughput hints at the end of the congestion, and the BWSS responds by setting σ to the new measured aggregate throughput. As shown in Section 4, this new reactive nature of the σ Calculation Sub-system allows faster convergence to the rate than the previous approach in [8].

4 Internet Experiments

To demonstrate the efficacy of video streaming over TCP using the BWSS, we have performed experiments using the Internet as our network testbed. We first discuss the experimental setup in more detail. We then describe an experiment which offers an example of a situation in which the BWSS offers better performance than TCP and a congestion-adaptive UDP protocol, using a popular video streaming application for our tests. Specifically, we compare the streaming performance of video encoded in SureStream RealVideo format streamed to a linux RealOne[13] client using TCP, TCP with the BWSS, and UDP.

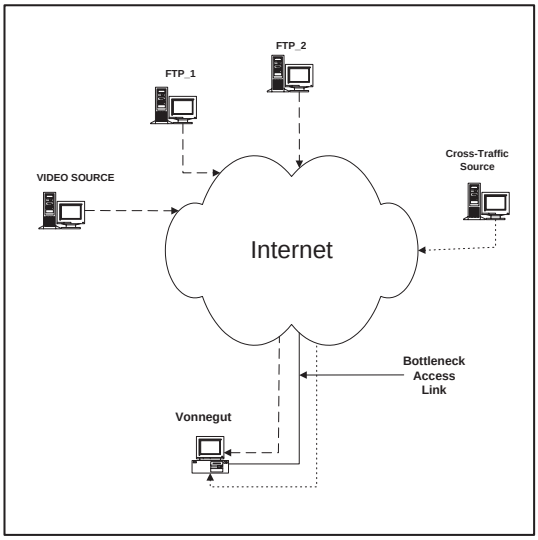

Figure 3: Experimental Setup. FTP/Video and interfering cross traffic sent to **Vonnegut** running BWSS

4.1 Experimental Setup

The network configuration used in our Internet experiments is shown in Figure 3. In the experiments, the host **Vonnegut**, a machine located in the eecs.berkeley.edu domain, is the recipient of Internet traffic. Since **Vonnegut** is actually connected to the Internet through a fast connection, we utilize the NIST Net [14] network emulation package to emulate a slower Broadband connection, which serves as our access link bottleneck. NIST Net employs a Linux kernel module to buffer incoming packets to limit the bandwidth of the connection, and to introduce delays which reflect a slower Internet connection. We use NIST Net to limit our overall incoming throughput to 960 Kbps and to introduce an additional delay of 30ms, which models the performance of a 1Mb/s Broadband Internet connection, such as that provided by DSL and Cable-Modem Internet Service Providers (ISPs).

All experiments involve a particular video source and two FTP sources sending data to **Vonnegut**, which is running the BWSS. At some point during each experiment, cross-traffic is sent from a different source to **Vonnegut**, creating congestion on the access-link. The interfering cross traffic is a constant bit-rate (CBR) UDP data stream generated

using the Real-Time UDP Data Emitter (RUDE) [15]. We use the throughput seen by the different applications, as well as the aggregate throughput of the receiving host as our performance criteria. The throughput received by the different applications is measured by a throughput measurement library which records the timestamps and sizes of packets read by the different applications. We perform 3 trials of each given experiment, and graph the average of these runs in all of our figures.

It is important to note that the BWSS is unable to control the interfering cross-traffic in any manner. The motivation for performing this sort of experiment is that it simulates the scenario when a single access-link to the Internet may be shared among multiple PCs connected via a LAN. In this case, it is possible for a user of one machine to generate traffic which causes congestion on the access-link and interferes with traffic destined to another machine on the LAN. It also models the case when a new UDP connection is started on the host running the BWSS.

4.2 RealVideo Streaming Experiments

We now demonstrate that streaming with TCP and the BWSS can offer better performance than either standard TCP or a congestion-adaptive UDP protocol. We conduct an experiment in which we stream a trailer for the movie "The Lion King" encoded using RealNetworks's SureStream [1] technology. SureStream technology, supported by RealNetworks's Helix Producer [16], supports encoding of the video stream at multiple bit-rates, and dynamically switches between the different encoded streams based on the available bandwidth. The trailer is encoded to support several different bit-rates: 450Kbps, 350Kbps, 262 Kbps, and 60Kbps. We choose to use SureStream technology since it is representative of the standard industry approach being taken by streaming media applications to address the congestion control deficiencies of UDP. Note that TCP-friendly streaming protocols must react to congestion by reducing the sending rate regardless of whether the congestion takes place in the network or at the user's access-link. The BWSS, on the other hand, is aware of the other TCP connections running on the user's host, and is able to break the fairness among these flows, without adversely affecting external flows, in order to obtain additional bandwidth for high-priority applications, such as video streaming.

We have installed a basic version of the Helix Universal Server [17] at the video source used for streaming the trailer. In addition to the streaming of the trailer, we have also executed two concurrent FTP sessions to ftp12.freebsd.org and ftp13.freebsd.org . At time 60 seconds, we introduce a 240Kbps interfering UDP flow which lasts for 40 seconds. As shown in Figure 4, the TCP SureStream flow is unable to maintain the necessary throughput for streaming at 450Kbps. We observe that with TCP, the SureStream technology does not try to switch to the 350Kbps encoding when the throughput decreases, resulting in a very poor viewing experience.

Meanwhile as Figure 5 shows, with UDP, the SureStream technology is capable of effectively streaming at 350Kbps [2], but does not switch to the 450Kbps stream until the congestion has subsided at approximately 100 seconds into the experiment. This is evidenced by the rise in throughput for the video stream at 110 seconds, and the switch from the 350Kbps stream to the 450Kbps stream is confirmed by the RealOne client. The throughput of the RealOne client using TCP and the BWSS is shown in Figure 6. The video streaming connection is assigned a higher priority than the FTP applications and has a minimum rate of 520Kbps [3] The remaining bandwidth is shared equally among the FTP connections which were assigned a weight of 1 in the BWSS. As shown in Figure 6, the BWSS is able to ensure that the RealOne client has enough bandwidth to effectively stream the trailer at 450Kbps, despite the introduction of the interfering cross-traffic. The average streaming rate for this experiment for TCP, UDP, and TCP with the BWSS are 354Kbps, 442Kbps, and 488Kbps respectively. These results confirm our initial claims that the BWSS can provide better streaming performance than standard TCP, and in certain situations, can actually provide superior performance to congestion-adaptive UDP protocols.

[1] SureStream, Helix Producer, Helix Universal Server and RealOne Player are trademarks of RealNetworks Inc.

[2] The Helix Universal Server streams video at a faster rate than its encoded-rate whenever possible, most likely to fill the receiver's buffer to deal with varying network conditions. This phenomenon is shown in Figure 5 where the server sends the 350Kbps encoded stream at 400Kbps

[3] Even though the highest encoded bit rate is 450Kbps, we choose 520Kbps to be the minimum rate in order to accomodate the Helix Universal Server

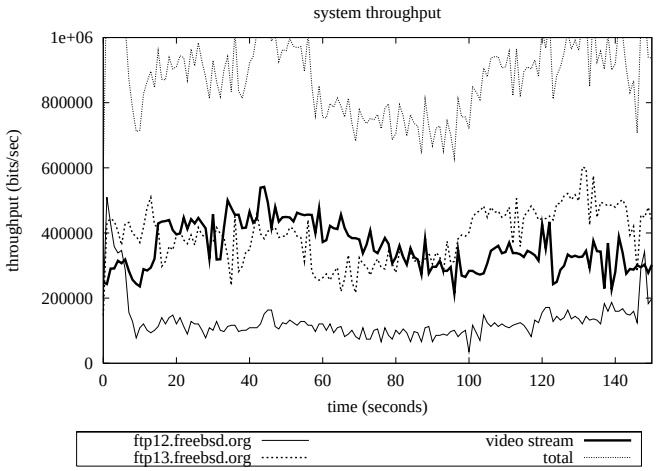

Figure 4: Bandwidth partition for TCP SureStream for experiment 2

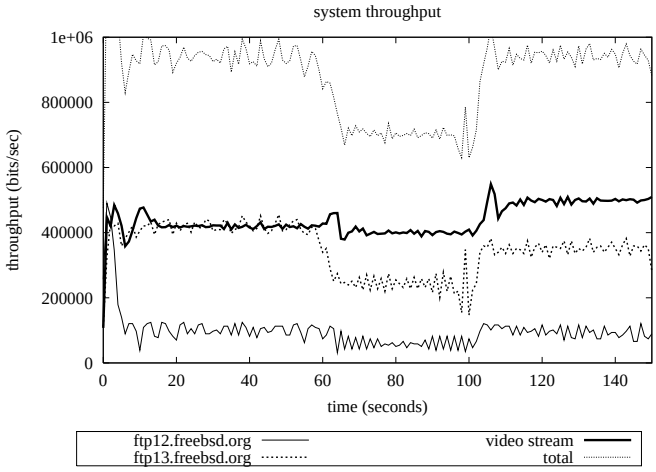

Figure 5: Bandwidth partition for UDP SureStream for experiment 2

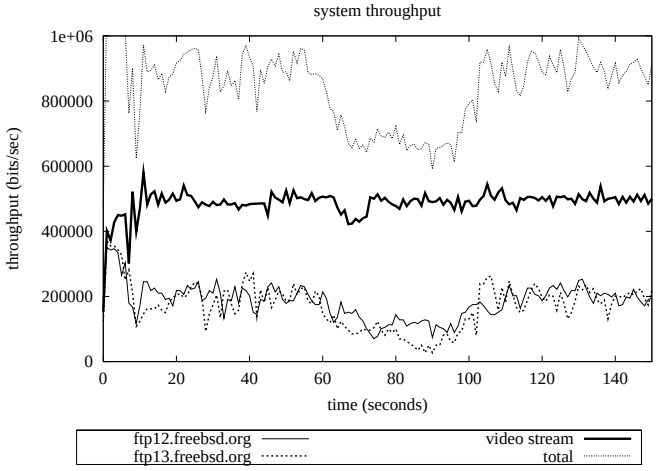

Figure 6: Bandwidth partition for BWSS SureStream for experiment 2

5 Conclusions and Future Work

In this work we have conducted a preliminary investigation of video streaming over TCP using a receiver-controlled BWSS. The BWSS allows prioritization of certain connections by providing them with additional bandwidth and works without changes to the network infrastructure or sending hosts. The BWSS achieves this prioritization by breaking the fairness among a user's TCP connections in a manner unavailable to TCP-friendly UDP protocols, which must ensure fairness with all competing TCP traffic, regardless of the destination. Through Internet experiments we have shown that streaming with the BWSS offers superior performance to streaming with standard TCP alone. Futhermore, we have demonstrated situations in which streaming using the BWSS can offer better performance than even congestion-adaptive UDP streaming protocols.

A natural question to ask is whether the BWSS can possibly be extended to incorporate both UDP and TCP flows. This future direction of research leads to interesting challenges. The most obvious is that although the BWSS sacrifices fairness among a user's TCP connections, it does not modify TCP in any manner that leads to unfairness with competing TCP traffic. Great care must be taken when operating with UDP flows to ensure that they remain TCP-friendly. One possible approach is to create a *virtual pipe* for the UDP traffic by restricting the TCP traffic to some fraction of the overall bandwidth. Hence it may be possible to set aside 500Kbps for a particular UDP streaming application and to restrict the user's competing TCP traffic to the remaining available bandwidth so that it does not interfere with the UDP stream. Additionally, while TCP offers an application-independent manner of controlling the bandwidth of certain flows by restricting the user's advertised window, UDP traffic is inherently application specific and offers no similar control knobs. We intend to explore different techniques to generalize the BWSS to provide a complete end-host solution to allow application-specific bandwidth allocation regardless of the underlying network protocol.

References

[1] "MovieFlix. http://www.movieflix.com," .

[2] J.B. Postel, "Transmission Control Protocol," RFC 793, Information Sciences Institute, September 1981.

[3] Van Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM '88*, Stanford, CA, August 1988, pp. 314–329.

[4] Sally Floyd and Kevin Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, 1999.

[5] Wai-tian Tan and Avideh Zakhor, "Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 172–186, 1999.

[6] Sally Floyd, Mark Handley, Jitendra Padhye and Jorg Widmer, "Equation-Based Congestoin Control for Unicast Applications," in *Proceedings of ACM SIGCOMM 2000*, August 2000.

[7] Reza Rejaie,Mark Handley and Deborah Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *Proceedings of IEEE INFOCOM 1999*, 1999, pp. 1337–1345.

[8] Puneet Mehra, Christophe De Vleeschouwer and Avideh Zakhor, "Receiver-Driven Bandwidth Sharing for TCP," in *Proceedings of IEEE INFOCOM 2003*, 2003.

[9] Charles Krasic, Kang Li and Jonathan Wapole, "The case for Streaming Multimedia with TCP," in *8th International Workshop on Interactive Distributed Mutlimedia Systems (iDMS)*, 2001.

[10] Pai-Hsiang Hsiao, H.T. Kung and Koan-Sin Tan, "Video over TCP with Receiver-based Delay Control," in *Proceedings of ACM NOSSDAV*, 2001.

[11] B. Mukherjee and T. Brecht, "Time-Lined TCP for the TCP-Friendly Delivery of Streaming Media," in *Proceedings of IEEE ICNP '00*, November 2000.

[12] Sam Liang and David Cheriton, "TCP-RTM: Using TCP for Real Time Applications," *Submitted to IEEE ICNP '02*, 2002.

[13] "RealOne Player http://www.real.com/," .

[14] "NIST Net network emulator. http://snad.ncsl.nist.gov/itg/nistnet/," .

[15] "Real-Time UDP Data Emitter (RUDE) http://cvs.atm.tut.fi/rude/," .

[16] "Helix Producer User's Guide http://service.real.com/help/library/guides/helixproducer/Producer.htm," .

[17] "Helix Universal Server Administration Guide http://service.real.com/help/library/guides/helixuniversalserver/realsrvr.htm," .