Receiver-Driven Bandwidth Sharing for TCP and its Application to Video Streaming

Puneet Mehra, Christophe De Vleeschouwer, and Avideh Zakhor, *Fellow, IEEE*

***Abstract*—Applications using Transmission Control Protocol (TCP), such as web-browsers, ftp, and various peer-to-peer (P2P) programs, dominate most of the Internet traffic today. In many cases, users have bandwidth-limited *last mile* connections to the Internet which 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 consequently may result in a bandwidth partition which does not necessarily coincide with the user's desires. In this work, we present a receiver-based bandwidth sharing system (BWSS) for allocating the capacity of last-hop access links according to user preferences. Our system does not require modifications to the TCP protocol, network infrastructure or sending hosts, making it easy to deploy. By breaking fairness between flows on the access link, the BWSS can limit the throughput fluctuations of high-priority applications. We utilize the BWSS to perform efficient video streaming over TCP to receivers with bandwidth-limited last mile connections. We demonstrate the effectiveness of our proposed system through Internet experiments.**

***Index Terms*—Bandwith allocation, multimedia streaming, TCP.**

I. Introduction

DESPITE THE recent explosion in availability of broadband Internet access, the majority of home users have relatively small-bandwidth links in comparison with the sites hosting desired content. For example, most ftp sites and websites are hosted on connections which can easily exceed 45 Mbps, while the fastest downstream residential access is generally limited to 1.5 Mbps. It is quite common for users to run multiple networking applications on a single connection, and the growing popularity of recent peer-to-peer (P2P) file-sharing services such as Napster [1], KaZaA [2], and Gnutella [3] has made this practice even more commonplace since they are usually left running for the duration of the Internet connection. Consequently there are many circumstances in which the last hop link becomes a bottleneck resulting in congestion among a user's applications.

Manuscript received May 7, 2003; revised December 1, 2003. This work was performed while C. De Vleeschouwer was a Visitor at the University of California at Berkeley, supported by the Belgian National Science Foundation. This work was supported by the National Science Foundation under Grant CCR-9979442 and the Air Force Office of Scientific Research under Contract F49620-00-1-0327. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ton A. C. M. Kalker.

P. Mehra and A. Zakhor are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: pmehra@eecs.berkeley.edu avz@eecs.berkeley.edu)

C. De Vleeschouwer is with the Laboratoire de Télécommunications, Université catholique de Louvain, Louvain-la-Neuve, Belgium (e-mail: devlees@tele.ucl.ac.be).

Digital Object Identifier 10.1109/TMM.2005.846783

The majority of traffic present on the Internet today is comprised of Transmission Control Protocol (TCP) [4] flows. Standard TCP does not provide any mechanisms for controlling the bandwidth allocated to a particular flow, and two connections which have the same round-trip time (RTT) generally receive an equal share of the bandwidth at a particular bottleneck link. This equitable sharing of bandwidth is desirable if the connections belong to different users of a network, but it may not maximize user satisfaction if the flows belong to the same user. It is conceivable for a user to want to prioritize different applications and distribute bandwidth according to his or her preferences. This is certainly the case when connections with different RTT co-exist, because TCP favors short RTT connections, which can receive a much larger share of bandwidth at a bottleneck link than flows with larger RTT [5].

A common form of bandwidth allocation is to allow weighted fair sharing of bandwidth among different applications. For instance, a user may decide to set aside one fourth of the available bandwidth for a P2P sharing application, another fourth of the bandwidth for an ftp download, and to allocate the remaining bandwidth for web browsing. However, there are cases in which an application requires a minimum guaranteed bandwidth allocation regardless of current link capacity. Multimedia streaming applications are a prime example of such applications, since they generally require constant playout at a particular rate, and are sensitive to fluctuations in the received rate. Many online games also have strict minimal bandwidth requirements for adequate usability. These applications can suffer from severe performance degradations if they fail to receive a minimum desired bit-rate. Hence it may be desirable to specify a minimum bit-rate for these applications regardless of the total link capacity, and to perform weighted sharing of any remaining bandwidth. This approach is the one envisioned in this paper.

In this work, we present an entirely receiver-based bandwidth sharing system (BWSS) which achieves the aforementioned prioritization and weighted sharing of bandwidth among a receiver's TCP flows with a common bottleneck. In most practical situations, this bottleneck is the last hop link for user access to the Internet. Our approach does not require any modifications to the network infrastructure or assistance from the sender. The proposed solution is fully compliant with standard TCP senders, and since it only requires receiver-side modifications, it is easily deployable. Our work is primarily focused on long-lived TCP connections such as file transfer and multimedia streaming. There are two main contributions of our work. The first one is a TCP Flow Control System (FCS) which achieves a particular target bit-rate for a given TCP connection, by controlling the receiver's advertised window. The second

contribution is a bandwidth-sharing system (BWSS), which uses the FCS to share the link bandwidth between different flows according to user preferences.

An additional interest of our work is to dispel the notion that TCP is always unsuitable for multimedia streaming. We focus on the common practice of streaming video to receivers whose *last mile* connections to the Internet are bandwidth limited. Since standard TCP shares bandwidth according to RTT, it may not provide streaming applications with the necessary bit-rate they need to function properly. We utilize the BWSS to address this shortcoming of TCP to perform efficient video streaming. The BWSS allows a user to break fairness among her own flows, and to partition bandwidth in an application-specific manner. Hence, the BWSS can be used to eliminate throughput fluctuations in TCP, which are detrimental for streaming applications. 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.

The rest of the paper is organized as follows. In Section II, we discuss relevant prior work. In Section III, we present an overview of our receiver-based system. In Section IV, we present a detailed description of the FCS for a single TCP connection. This control system is used by the BWSS discussed in Section V. In Section VI, we present a Linux implementation of our proposed system and the results of various Internet experiments. We discuss areas for further investigation and conclude this paper in Section VII.

II. Related Work

There has been extensive research in Fair Queuing [6] scheduling policies to allow bandwidth allocation at routers [7], [8]. PacketShaper [9] is a hardware solution which can provide bandwidth allocation and management for service providers. However, these solutions all require changes to the network infrastructure, and thus have not seen widespread deployment. The use of these mechanisms to support individual user preferences would result in additional state at the routers, leading to scalability problems. Updating these preferences would also result in additional router management issues. Our BWSS achieves prioritization and weighted sharing of bandwidth among a receiver's TCP flows by controlling the receiver's advertised window without any modifications to the network infrastructure.

There has been some prior work on providing service differentiation for TCP flows through modification of either the advertised window or the delay in ACK packets. The authors of [10] share our goals of providing an entirely receiver-based mechanism to prioritize TCP traffic, and they leverage the receiver's advertised window in order to achieve differentiation. Their work primarily focuses on reducing queuing delays at the receiver for interactive applications, and on providing more bandwidth for short data transfers all at the expense of long-lived flows. Their proposed scheme allocates a minimal window of 1 to all long-lived flows, which works on low-bandwidth modem links, but may limit throughput on faster connections. Our approach differs because our goal is to achieve a desired weighted bandwidth partition, as well as to target a minimal rate for certain applications. Furthermore, our system adjusts to congestion, while their approach requires explicit knowledge of the link capacity in order to allocate buffer sizes. The authors of [11] propose adjusting the receiver's advertised window at a web cache to achieve proportional fairness among flows. Their work does not adapt the window to congestion and does not examine the time scale needed for accurate bandwidth estimation. The authors in [12] use the advertised window to limit the rate of TCP video traffic on a VPN link between a video server and proxy servers. Authors in [13] propose delaying TCP ACK messages at the endpoints of a connection in order to reduce bandwidth. The fundamental goal is to reduce congestion related queuing and timeouts at routers in order to support streaming applications.

The Congestion Manager (CM) [14] framework is similar to our proposed BWSS in that it seeks to share available bandwidth among a set of flows. However, while our proposed BWSS runs on a receiving host, the CM is designed to run on the sender side. The sending host takes priorities into account to allocate bandwidth among flows that have the same destination. Thus, the allocation of resources is performed between the set of flows accessing the same given server. Meanwhile, our system considers bandwidth allocation among all the flows reaching a given receiver, regardless of the senders. Typically, it allocates resources among different applications, independently of the accessed server.

Another focus of our work is the possibility of efficient video streaming over TCP. There have been several recent proposals which challenge the long-held belief that TCP is unsuitable for streaming applications. The authors of [15] 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) [16], in which receivers delay TCP ACK packets based on router feedback. The authors of [16] mimic a constant bit rate (CBR) connection using RDC and also propose a layered streaming method. In contrast, our approach leverages our proposed BWSS to provide a nearly CBR connection for the video stream, without any changes to routers or sending hosts. Time-lined TCP [17] 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 [18] 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 Time-lined TCP or TCP-RTM to provide guaranteed throughput for

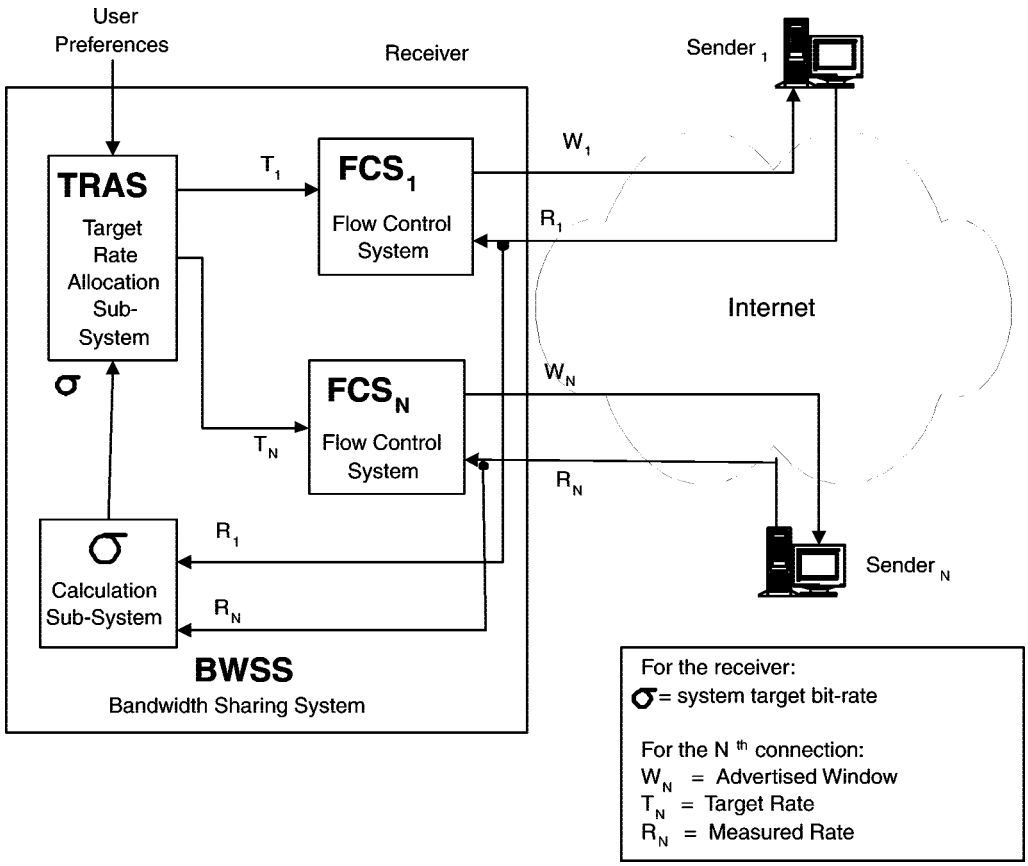

Fig. 1. Receiver-based system for bandwidth sharing (BWSS).

a TCP connection, in addition to the real-time performance enhancements of these protocols.

III. System Overview

We now provide an overview of our receiver-based solution for sharing link bandwidth according to user preferences. There are two objectives of our proposed system: to achieve full utilization of the receiver's access link, and to satisfy user preferences regarding how the bottleneck bandwidth should be shared among different applications. A formal definition of these goals is provided in Section V. Since we do not assume any a priori knowledge of the receiver's link capacity, the *full utilization* of the link simply refers to the aggregate throughput achievable by the flows when operating under standard TCP. Our goal is to use as much of the link bandwidth as standard TCP, but in a distribution which matches user preferences. The essential idea behind our system is to constrain the throughput of certain low-priority flows to provide additional bandwidth, if possible, for higher-priority flows as specified by the user's profile. It is important to note that although our system breaks fairness among a receiver's TCP connections, it is still fair to competing TCP traffic from other users. This is because the receiver's advertised window can only constrain the sender's congestion window.

A block-diagram of our proposed system is shown in Fig. 1. The main building block of our system is the FCS, which can constrain the rate of a given TCP connection to a particular target bit-rate. As shown in Fig. 2, the input to the FCS is the desired target bit-rate, while the output is the required advertised window to achieve the desired target rate. The FCS is an iterative three stage process which consists of measuring the actual bit-rate of a flow, calculating the difference between the actual and target bit-rates, and then adapting the receiver's advertised window to achieve the desired target bit-rate. As we describe in Section IV-B, both the period of adjustment and the time scale used to measure the rate depend on the estimated round trip time of the connection. Given the FCS, a naive approach to the problem would be to simply measure the maximum achievable receiver link bandwidth, and to calculate the target bit-rates for the different FCSs according to the weights assigned to each application by the user. While this approach would certainly achieve the goal of matching user preferences, it may not necessarily achieve full link utilization since certain flows might be limited by network bottlenecks, maintaining their throughput below the desired target bit-rate. Such network bottlenecks motivate the need for our proposed BWSS, which determines the appropriate target bit-rates for the FCSs, based on user preferences and network measurements, to ensure full utilization of the receiver's access link.

The overall working of our system can be described as follows. Assuming N TCP flows, their actual rates are measured and input to the "σ Calculation Subsystem," which determines the system target bit rate, denoted by σ. In essence, σ represents the sum of target bit-rates allocated to different flows. As shown in Fig. 1, given σ and the external user-preferences, the Target

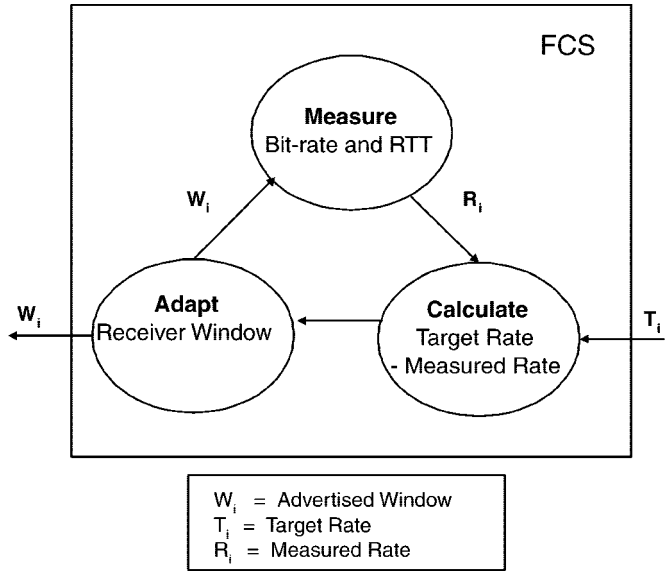

Fig. 2. TCP flow control system (FCS).

Rate Allocation Subsystem, further described in Section V-A, determines the target bit-rates for the different FCSs. 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, which is simply the sum of the actual rates. The σ Calculation Subsystem measures the actual aggregate throughput of the connections in order to determine the optimal value of σ, which achieves full link utilization. Specifically, the system uses increases and decreases in the aggregate throughput to guide corresponding increases and decreases in σ. These calculation mechanisms are explained in more detail in Section V-B. We now describe each component of our current BWSS implementation, beginning with the FCS.

IV. TCP Flow Control System (FCS)

The FCS aims to maintain a particular bit-rate for a given TCP flow. Its input is the desired target bit-rate, while the output is the TCP advertised window needed to achieve the desired target rate. The system is only effective if the desired rate is achievable under the flow's TCP congestion window. Our proposed system takes a target bit-rate as input, and measures the bandwidth and RTT of the flow. It continuously adapts the receiver's advertised window to achieve this desired target rate. We now present the algorithm used by the FCS to achieve the desired target bit-rate.

A. FCS Algorithm Description

The goal of the regulation process is to bring the actual rate R of the flow to within a fraction α of the target rate T, i.e., $R \in [(1-\alpha)T, (1+\alpha)T]$. We will refer to α as the *rate slack*, and the corresponding interval for R as the *desired interval*. The regulation process employs the receiver's advertised window, w, to control R. Our algorithm assumes an advertised window expressed as an integer number of packets, and packets of equal size, p_{size}. In the actual implementation, the advertised window is simply an integer multiple of the advertised TCP maximum segment size(MSS), and most packets are usually one MSS in size. Another constraint on our system is that the window value must be greater than zero, which follows from the fact that we must ensure that some data is transferred during each RTT.

Let p_{size} be the size, in bits, of each packet sent, and *RTT* be the estimated average round-trip-time for the flow. We will discuss exactly how *RTT* is estimated in the next section. Given these parameters, our proposed regulation process assumes the following model for the rate R of a TCP flow:

$$R = \frac{w \cdot p_{size}}{RTT}. \tag{1}$$

The initialization stage follows from this model. Let T be the desired target rate for the flow. The regulation process begins by setting w to $(T \cdot RTT)/p_{size}$, i.e., $w \leftarrow (T \cdot RTT)/p_{size}$. The regulation process then proceeds in an iterative manner with the estimated bandwidth measurement serving as a guide for further refinements in the advertised window. These refinements require a knowledge of the impact on R of a given change in w. The changes in R due to a change in w is calculated by differentiating (1), giving

$$\frac{\Delta R}{\Delta w} = \frac{p_{size}}{RTT}. \tag{2}$$

Before describing the steps of the FCS algorithm, we now outline its strategy. The goal is that, after a rapid initial convergence, the FCS progressively stabilizes around the desired rate. In practice, the FCS rapidly enters a state in which $R < T$. Then the advertised window is progressively increased, until R is in the desired interval. As detailed in Fig. 3, the FCS works as follows:

1) If the measured rate R is within a fraction α, the *rate slack*, of the desired rate T, i.e., $R \in [(1-\alpha)T, (1+\alpha)T]$, then no action is taken.
2) If $R > (1+\alpha)T$, then the system has to reduce R by decreasing w, subject to the constraint that $w \geq 1$. Hence, **(2)** is used to compute the necessary advertised window decrease $\Delta w'$, i.e.,

$$\Delta w' = \left\lfloor \frac{R-T}{\frac{\Delta R}{\Delta w}} + 0.5 \right\rfloor = \left\lfloor \frac{R-T}{\frac{p_{size}}{RTT}} + 0.5 \right\rfloor. \tag{3}$$

This process continues until either T is achieved or the rate R falls below the target T.
3) If $R < (1-\alpha)T$, then we increase w using **(2)** to determine the Δw which will lead to the desired ΔR. To stabilize the system, we limit the amount of change in w that can occur during any single adjustment by multiplying Δw by β, the *stability factor*, which must be less than one and has been set to 0.7 in our experiments. Hence, incorporating the stability factor, we have

$$\Delta w'' = \left\lfloor \beta \cdot \frac{T-R}{\frac{\Delta R}{\Delta w}} \right\rfloor = \left\lfloor \beta \cdot \frac{T-R}{\frac{p_{size}}{RTT}} \right\rfloor. \tag{4}$$

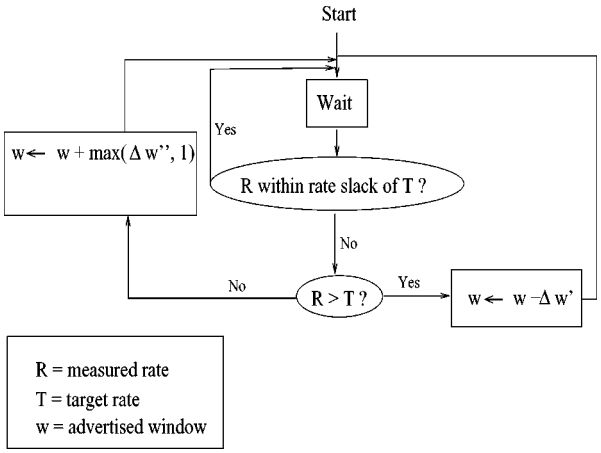

Fig. 3. TCP flow control system (FCS) algorithm.

Since we do not want to stay in a state for which $R < (1-\alpha)T$, the value of w is increased by at least one. In other words, w is increased by $\max(\Delta w'', 1)$.

After any change in the advertised window, the system waits for the change to have an impact on the throughput, before performing a new adjustment. An estimate of the RTT and an accurate measurement of flow bandwidth are also important for reasonable performance of the FCS. These issues are discussed in detail in the following section.

B. Measuring Flow RTT and Bandwidth

To calculate the RTT of each packet at the receiver, we employ the TCP Timestamp option [19]. Our proposed system uses the TCP smoothed RTT value, s_rtt, as an estimate of the average RTT for a given flow.

Accurate bandwidth estimation is a crucial component to ensure convergence and stability of the FCS, and is dependent on RTT as outlined below. Our system estimates R at the end of successive bandwidth-estimation periods, denoted by ϕ.[1] The minimum possible value for ϕ is one RTT, which would ensure fast convergence. However, since RTT may fluctuate and we want to ensure a reliable measurement of R, we set ϕ to twice the RTT value. The estimation of R relies on an exponentially-weighted moving average, which works as follows. Let R_ϕ be the number of bytes received per unit of time over the bandwidth-estimation period ϕ. Let ϵ be the exponential average parameter; the rate estimation is adjusted as follows: $R \leftarrow \epsilon \cdot R + (1-\epsilon)R_\phi$.[2] After an adjustment of the advertised window, past estimations of R become irrelevant. In that case a weighted average makes no sense, and R is simply set to R_ϕ. The new bandwidth-estimation period begins when the advertised window adjustment becomes effective, i.e., at least one RTT after window adjustment at the receiver.

The last major characteristic of the FCS that needs to be discussed is the frequency of adjustment of the advertised window. After an adjustment, we have to wait for the change to be effective before considering the next adjustment. The time to wait, φ, is bounded based on two observations. First of all, the last change has to take effect in the system, which requires that φ be greater than one round trip time. Second, the bandwidth measurement has to be relevant. So, the minimum value for φ is dictated by $\varphi > RTT + \phi$. However, in order to ensure a reliable throughput estimation we set $\varphi > RTT + 3 \cdot \phi$. By chosing this value we allow enough time for the window adjustment to have an effect on throughput, and on a long enough period to compute a reliable average throughput. This latency inherent in the operation of the FCS is actually quite beneficial with respect to the BWSS. As will be further explained in Section V-C, the fact that there is some lag in the FCS response to changing network conditions allows the BWSS to determine the beginning and end of network congestion.

[1] In all experiments, $\phi = 2 \cdot RTT$.

[2] In our experiments $\epsilon = 0.3$, though the exact value is not important

V. Bandwidth Sharing System (BWSS)

In Section IV, we have shown that a TCP receiver is able to limit the throughput of a connection below its regular TCP rate. In this section, we consider the case of several TCP flows terminating at the same receiver, which share a common bottleneck. Our purpose is to allocate the bottleneck bandwidth among the contending TCP flows according to the receiver's preferences. The principle behind the system consists of constraining some flows in order to improve the throughput of others based on prespecified user preferences. Obviously, constraining a flow's throughput is only warranted if the receiver's other flows are able to take advantage of the constraint. The goal of our proposed system is to match user preferences while ensuring full utilization of the receiver's link capacity. In the following sections we first formalize the receiver's preferences in terms of priorities among the flows. Our proposed system, outlined in Section V-B, guarantees an optimal partition of the bandwidth within a bottleneck shared by all of the receiver's TCP connections. Typically, this situation is encountered when the last hop link to the receiver is the bottleneck. This system does not guarantee an appropriate partition of bandwidth for a bottleneck shared by only a strict subset of the flows. However, in such a bottleneck, a flow destined for a given receiver is likely to be aggregated with a large number of flows belonging to other users, and hence any bandwidth made available by constraining a flow is shared among these competing flows. To deal with cases where a strict subset of the receiver flows share a bottleneck with little or no external aggregation, we have devised an alternative bandwidth sharing system, but omit the discussion of that more complex system due to its longer time for convergence, and to the fact that such a bottlenecks are unlikely in a real network.

A. Target Rate Allocation and Receiver Preferences

Receiver expectations for a TCP flow are likely to depend on the application using the connection. For example, streaming applications are only viable above some bandwidth threshold, and for a given precoded content at a fixed bit-rate, the perceptual quality does not significantly improve when the TCP bandwidth increases. Meanwhile, a file transfer application does not

have a strict minimal bandwidth requirement, but benefits 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. These parameters relate the receiver expectations in terms of bandwidth allocated to each connection. First, the minimal rate should be provided to every connection, in decreasing order of priority. Then, the remaining bandwidth should be 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.

We have previously outlined how the FCS can achieve a desired bit-rate for a given connection. We refer to the throughput which the FCS aims to achieve, by adapting the receiver's advertised window, as the *target bit-rate* for a flow, or simply the *target rate*, T. As mentioned in Section III, the Target Rate Allocation Subsystem (TRAS) uses the system target bit-rate σ, along with user preferences, to determine the target bit-rate for each FCS in the BWSS. The value σ represents the total target bit-rate allocation for the entire system, which must be distributed among the different flows in the system. σ is set by the σ Calculation Subsystem, discussed later in Section V-B, which takes actual network measurements into account in determining its value. Let us assume there are N flows in the bandwidth-sharing system, and let T_i be the target rate of the i^{th} flow. For the set of N flows, denote $\{T_i\}_{0\leq i<N}$ to be the set of target rates. Once σ is set by the σ Calculation Subsystem, it is used by the TRAS to derive the target bit-rates, T_i, for each of the flows, subject to the constraint

$$\sum_{i=0}^{N-1} T_i = \sigma. \tag{5}$$

Note that there are many $\{T_i\}$ which satisfy (5), and we refer to any such set of target bit-rates for the flows destined to the receiver as a *receiver partition*. We now introduce the concept of a *desired partition*. A desired partition, much like a receiver partition, refers to the target rates and not to the actual rates. Intuitively, a desired partition represents the receiver partition for a given σ, which also matches user preferences. While there are many possible receiver partitions for a given σ, there is a unique desired partition. The TRAS calculates the desired partition corresponding to its input σ. We now provide a formal definition of the desired partition. Let p_i, m_i, and w_i be the priority, the minimal rate, and the weight of the i^{th} flow, respectively. The definition of the desired partition depends on whether all flows have been provided with their minimal rate.

In the first case, we assume that the minimal rate has been satisfied for all flows and thus $\sigma \geq \sum_{j=0}^{N-1} m_j$. In this case, any *remaining* bandwidth should be shared among the flows according to their respective weights. More formally, in this case, $\{T_i\}_{0\leq i<N}$ is a desired partition if $\forall i$

$$T_i = m_i + w_i \cdot \frac{\sigma - \sum_{j=0}^{N-1} m_j}{\sum_{j=0}^{N-1} w_j}. \tag{6}$$

Note that $(\sigma - \sum_{j=0}^{N-1} m_j)$ represents the amount of remaining bandwidth after all flows have been provided with their minimal requirements. Furthermore, $w_i / \sum_{j=0}^{N-1} w_j$ represents the amount of this remaining bandwidth which should be additionally allocated to the i^{th} connection.

In the second case, we assume the minimal bandwidth requirements cannot be met for all flows since $\sigma < \sum_j m_j$. Then the bandwidth is allocated in decreasing order of priority and up to the minimal rate of each flow. There will be certain flows which receive less than their minimal rate, and some may even be completely starved. Let us assume that a flow with a larger priority value has a higher priority, and that the flows are arranged in strictly decreasing priority, thus $p_i > p_j, \forall i < j$. Then $\{T_i\}_{0\leq i<N}$ is the desired partition if $\forall i$

$$T_i = \min\left(m_i, \max\left(0, \sigma - \sum_{j=0}^{i-1} m_j\right)\right). \tag{7}$$

The formulation in (7) states that in the desired partition, each flow will have its minimal rate met in accordance with its priority. Thus the most important flow will have its minimal requirement met, if possible, and any remaining bandwidth will be allocated to the next highest priority flow, continuing until the entire sum, σ, has been utilized. One flow will have a target rate between 0 and its minimal rate, while the remaining flows of lesser priority will have a target rate of 0 assigned by the system.

We conclude this discussion with the concept of an optimal partition. An *optimal partition* is a set of target rates together with σ, resulting in actual rates that best fit the receiver preferences while fully utilizing the receiver's link capacity.

B. σ Calculation Subsystem Overview

Before delving into the details of the σ Calculation Subsystem, we provide a brief recap of our receiver-based system, depicted in Fig. 1. Recall that for a given σ, the target rate for each flow i, T_i, is computed by the Target Rate Allocation Subsystem (TRAS). These T_i are input into the appropriate FCS for the different flows, which compute the advertised window values to achieve T_i. Let R_i be the actual measured bit-rate of the i^{th} flow. Furthermore, let the link utilization, U, be the sum of the measured actual bit-rates, i.e., let $U = \sum_i R_i$. Since the T_i are computed by the TRAS, they always match user preferences. Thus the main goal of the σ Calculation Subsystem is to choose σ to fully utilize the link capacity, i.e., to maximize U.

The relationship between σ and the resulting values of link utilization, U, is shown qualitatively in Fig. 4. As seen, for small

values of σ, U grows monotonically as σ increases. However once the link is fully utilized, as depicted by *link capacity* in Fig. 4, further increases in σ will not result in an increase in link utilization. In Fig. 4 we denote σ_{ideal} to correspond to a value of σ resulting in the set of actual rates which best matches user preferences, subject to the constraint that U is maximized, i.e., subject to the constraint that we have full utilization of the link. As shown in Fig. 4, choosing a $\sigma < \sigma_{ideal}$ results in under-utilization of the link. While selecting $\sigma > \sigma_{ideal}$ does not result in improved link utilization, since U is maximized for this value; rather it causes the system to relax constraints placed on lower priority flows, and causes the system to degenerate toward classical TCP operation in which priority is determined by a flow's RTT rather than user preferences. Given these observations regarding the effects of modifying σ, we can now more precisely restate the goal of the σ Calculation Subsystem. The goal is to find σ_{ideal}, the smallest σ, which achieves full link utilization. After initial convergence to the optimal σ value, the σ Calculation Subsystem is passive and only changes σ when network conditions change. The basic approach used by the σ Calculation Subsystem to achieve σ_{ideal} is to use measured increases and decreases in U to guide corresponding increases and decreases in σ. We discuss this process, and the guiding rationale behind it, in further detail in the following section.

C. Reactive σ Calculation Subsystem Details

Upon startup, the BWSS enters an initialization phase in which it estimates the total available receiver bandwidth by releasing the constraints on all the flows and setting σ to this value. In other words, σ is initialized to the aggregate throughput of all flows under regular TCP, without the BWSS in effect. Since some of the flows might be limited by network bottlenecks, this initial partition does not necessarily provide full link utilization. Consequently, the system progressively increases σ until further increases of σ do not result in any corresponding increases in the aggregate throughput of the connections, at which point full link utilization has been achieved. This initialization phase ensures convergence toward full link utilization while matching user preferences for the bandwidth partition. After this step, σ has achieved its optimal value for the current network conditions. The goal of the reactive σ Calculation Subsystem is then to adapt the σ value to changing network conditions. To cope with the possibility that the reactive system misses a change in network conditions, periodic reinitialization of the system is performed to guarantee convergence on a long time scale.

The design of the reactive system relies on two fundamental observations: first, when congestion affects a flow, the measured throughput of the flow decreases; second, when congestion subsides, the measured throughput increases. The latter statement is true even if the FCS controls the throughput of a particular flow. Congestion for a given flow is marked by an increase in the router queues along the path to the sending host, resulting in a longer RTT and consequently smaller measured throughput. When congestion subsides, these router queues drain, resulting in a decrease in the affected flow's RTT and consequently an increase in the measured throughput for the flow. Hence, the measured throughput increases whenever congestion subsides,

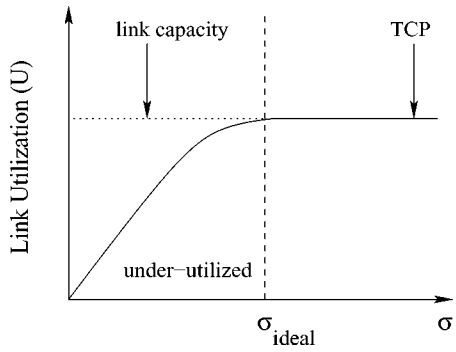

Fig. 4. Link utilization as a function of σ.

even if the sender window does not change. Similarly, the measured throughput decreases at the onset of congestion, even if the window is held constant. The inherent latency of the FCS, i.e., the fact that the FCS cannot instanteously adjust the window and hence flow throughput to match the target value, allows the reactive σ calculation system to detect changing network conditions.

Assuming that the initial σ value is optimal for the initial network conditions, the previous observations about congestion have been used to guide the increase and decrease of σ when congestion appears or subsides in the network. 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. The system must be able to distinguish these two cases, and respond accordingly 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, γ, of its target bit-rate, T^*, then σ is **increased** by $(1-\gamma)\cdot T^*$.[3] The BWSS keeps track of a flow that experiences congestion, and when the measured throughput increases and becomes larger than $\gamma \cdot T^*$, 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 $\gamma \cdot T^*$, 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.

The reactive system makes the assumption that the σ value is optimal for the current network conditions, and provides a mechanism able to adapt σ as a function of instantaneous changes in the measured throughput of receiver flows. However, some of these changes may only be observed for a short period

[3]In our experiments, $\gamma = .8$

of time after a change in network conditions. For example, when congestion affecting a particular flow subsides, there is a measurable increase in the throughput due to the decrease in RTT. However, after some time, the FCS is able to adjust the advertised window to reduce the flow's throughput to its target rate. Hence, if the system misses the initial instantaneous increase in the flow's throughput, then it might remain in a suboptimal state. The periodic reset of the BWSS is needed to ensure that the system does not stay in such a suboptimal state on a long time scale. The purpose of the re-initialization is to ensure long term convergence by re-synchronizing the system to the optimal σ value, in case the reactive system did not appropriately handle changing network conditions. A reset of the BWSS causes the system to enter the initialization phase described in Section V-C. During the initialization phase the BWSS releases the constraints on the connections and progressively increases σ until full link utilization has been achieved. The failure of the reactive system cannot be completely avoided due to biased bandwidth measurements for receiver flows caused by inherent loss-related TCP fluctuations.

D. Evolution of the BWSS

An initial version of our proposed system was presented in [20]. That version of the BWSS was tested using both NS-2 simulations [21] and real Internet experiments. Additional operational experience with the initial BWSS implementation over the Internet provided valuable lessons which guided our current implementation. Specifically, our current implementation has abandoned much of the complexity of our original prototype in [20] and provides better performance in addition to its simpler design. We now highlight some of the differences and the rationale behind these changes in order to provide the reader with a more complete picture of the approaches that have been investigated.

The earlier version of our system in [20] used an FCS which utilized delay in TCP ACK packets, in addition to modifying the receiver's advertised window, to control the throughput of a given connection. Further simulations revealed that adding delay was only useful in rare circumstances, e.g., in cases where there was a high degree of required precision in achieving the target bit rate. In addition, due to the constraints of our user-space Linux implementation, it was not possible to delay TCP ACK packets in Internet experiments. Our discussion of the FCS in this paper does not incorporate delay of ACK packets due to the limited utility of this feature.

In addition, the previous σ Calculation Subsystem proposed in [20] converged to the optimal value for σ by iteratively increasing and decreasing σ, and measuring the impact of these changes on the actual measured throughput of different connections. While this iterative approach demonstrated convergence for a wide range of network topologies and scenarios, it suffered from a long convergence time, limiting its utility for multimedia streaming applications. The σ Calculation Subsystem described in this paper is able to adapt to changes in flow throughput much more rapidly, and hence is better suited for our target applications.

VI. Internet Experiments

In this section, we present results obtained with the prototype of our proposed flow control system (FCS) and bandwidth sharing system (BWSS), implemented for the Linux operating system. We first highlight relevant implementation details and then present experiments using the prototype over the actual Internet, both for simple TCP bandwidth partitioning and for realistic video streaming scenarios. Note that all the graphs provided in this section have been drawn on a short time scale. This is to highlight the fast reaction capabilities of our system. A consequence of the use of a short time scale is that the periodic reset, performed on a larger time scale, typically a few minutes, only appears in the form of the initialization phase at system startup.

A. Linux BWSS Implementation Issues

Both the FCS and the BWSS have been implemented as a shared library, libvstcp. In libvstcp we override the *connect()* and *read()* functions from the C standard library, libc, in order to provide the desired functionality of the FCS and BWSS. We provide more detail about the exact changes in these functions later in this section. Any application using the BWSS must preload libvstcp before libc using the **LD_PRELOAD** environment variable. Since the BWSS must maintain state information for all of the connections in the system to operate correctly, we share state information between the different library instances, which are loaded by the networking applications, using the shared memory inter-process communication (IPC) facilities provided in linux. As previously stated, libvstcp overrides two libc functions: *connect()* and *read()*. In the *connect()* function, any TCP application obtains a pointer to the BWSS data structure and "registers" itself with the system using its process ID (PID). The registration process involves initializing data for this new connection and returns a pointer to the FCS data structure which has been allocated for this connection. The libvstcp *read()* function calls the libc *read()* function in order to determine the number of bytes which would be returned to the calling application. This measurement is used to update the bandwidth calculations for the FCS and to make any needed changes to the advertised window in order to meet the desired target bit-rate. The *setsockopt()* system call is used to set the receiver socket buffer size, and hence the advertised window, to an integral multiple of the TCP advertised maximum segment size (MSS). Thus the FCS rate adjustment process for the application is carried out during the *read()* function call. The different phases of the BWSS, such as increasing the system target bit-rate σ, are also carried out during the *read()* function call. Due to the sharing of state information using the shared memory segment, it is possible to make a change in the BWSS in one application and observe the impact of the change during the *read()* function call in another application. We use the smoothed TCP RTT estimate, s_rtt, as our estimate of the average RTT for the FCS. The 2.4.x version of the linux kernel allows a user-space application to obtain the s_rtt associated with a TCP socket using the *getsockopt()* system call with the **TCP_INFO** parameter. In certain cases when the traffic is primarily one-way, as is the case in some ftp connections, s_rtt may be unavailable. Thus we also obtain an RTT estimate

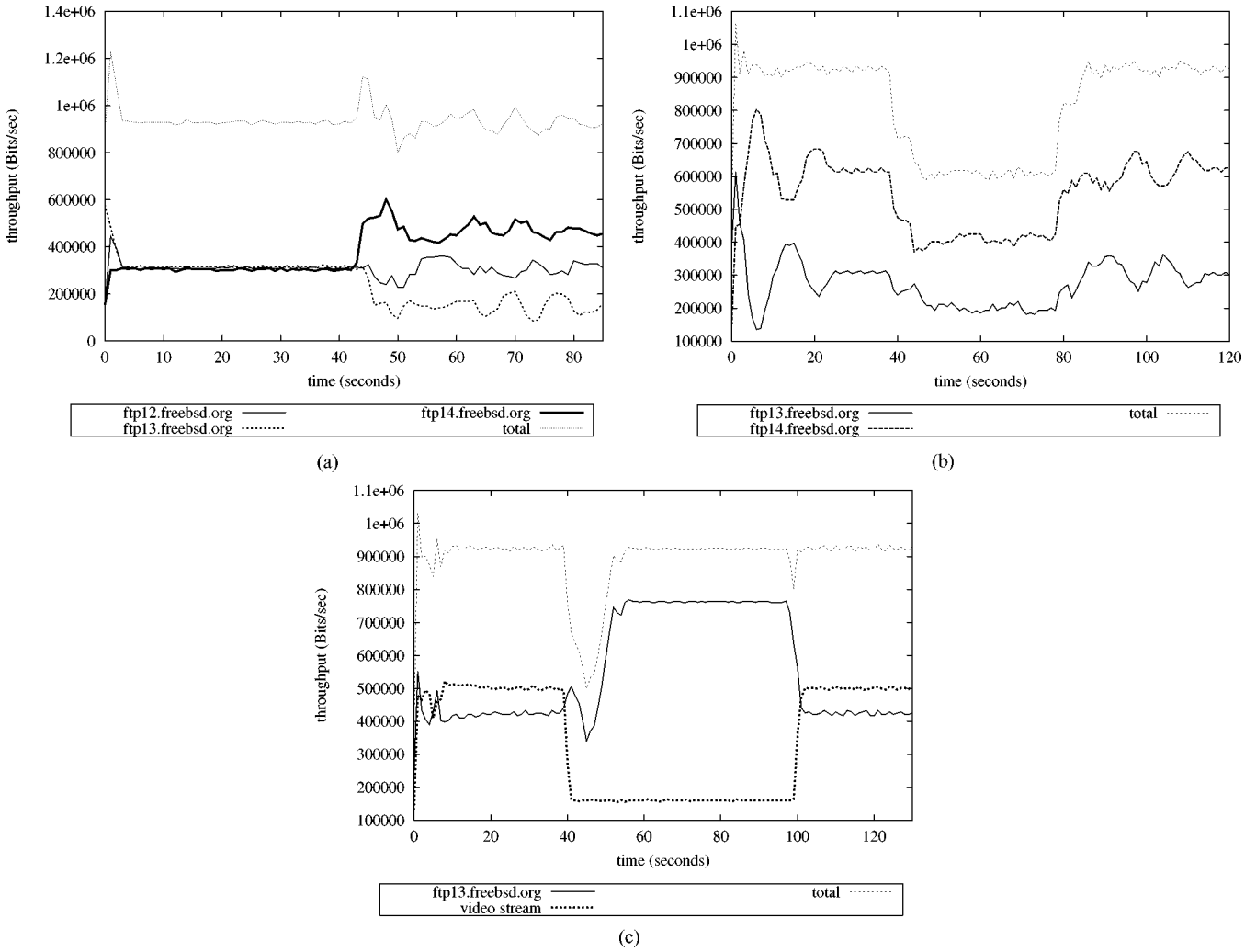

Fig. 5. Bandwidth partition for Internet experiments with BWSS. Results are obtained by averaging three runs. (a) Experiment 1 demonstrating the ideal case. (b) Experiment 2 demonstrating link bandwidth reduction at time 40; (c) Experiment 3 demonstrating bandwidth redistribution.

using the fping [22] program, which relies on Internet Control Message Protocol (ICMP) echo requests.

B. Methodology and Results for BWSS Experiments

We now present the methodology and results of experiments conducted with actual Internet hosts designed to demonstrate the operation of the BWSS under a variety of conditions. The experiments consist of FTP downloads of FreeBSD ISO images from the different ftp servers. The receiving host running the BWSS, **Vonnegut** is a PC workstation running Mandrake Linux 8.1 with kernel version 2.4.8–26, located in the eecs.berkeley.edu domain. Since **Vonnegut** is actually connected to the Internet through a fast connection, we utilize the NIST Net [23] 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 30 ms, which models the performance of a 1 Mb/s Broadband Internet connection, such as that provided by DSL and Cable-Modem Internet Service Providers (ISPs).

We use the throughput seen by the different applications, as well as the aggegate throughput 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 three trials of each experiment, and graph the average of these runs in all of our figures.

1) Experiment 1 – The Ideal Case: This experiment demonstrates the ability of our system to appropriately allocate bandwidth to flows according to user preferences in scenarios with nearly constant access link bandwidth. We assign weights of 1, 2, and 3 to hosts ftp13.freebsd.org, ftp12.freebsd.org, and ftp14.freebsd.org respectively, and do not assign any minimum rate, resulting in a bandwidth distribution according to weight. During the initial 40 s, we allow standard TCP operation, and as shown in Fig. 5(a), all flows have the same measured throughput. At time 40 s, we start the BWSS, and as shown in figure, the prototype achieves the expected weighted flow throughput.

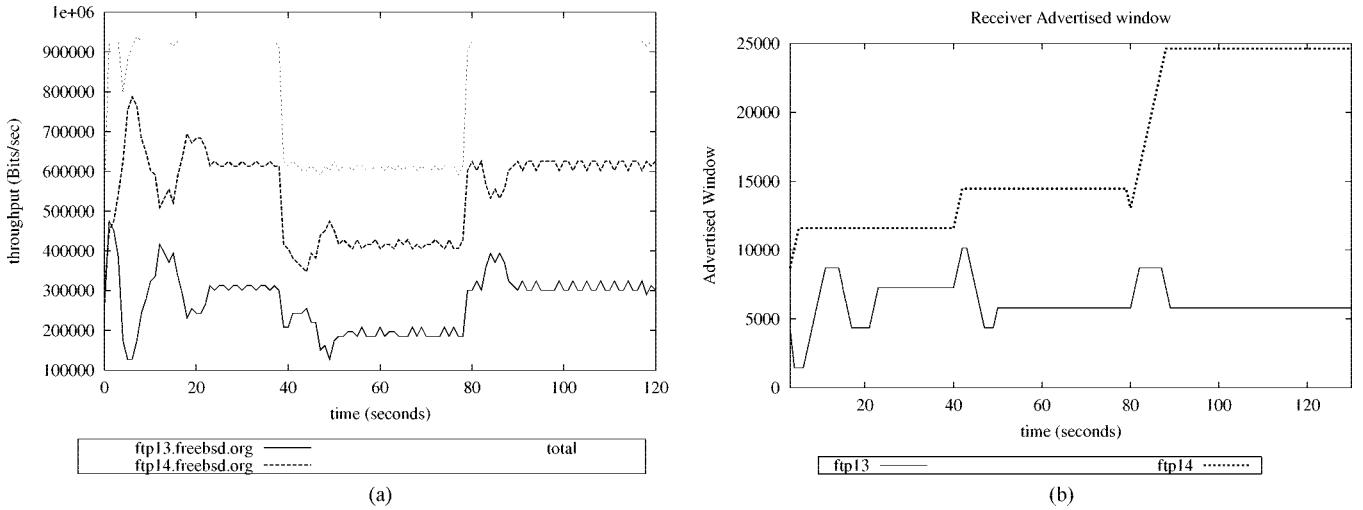

Fig. 6. Sample run for Experiment 2: (a) BWSS bandwidth partition and (b) corresponding advertised window.

2) Experiment 2 – Link Bandwidth Reduction: The following experiment investigates the ability of the BWSS to adapt to an overall reduction in the link capacity. This is accomplished by generating UDP cross traffic which is not under the control of the system. We assign weights of 1 and 2 to hosts ftp13.freebsd.org and ftp14.freebsd.org, respectively. No minimal rate is set, so the flow priority is not relevant. At time 40 s, a 320 Kbps UDP stream, terminating at the receiver host is started from another host located in the eecs.berkeley.edu domain. The UDP traffic, generated using the Real-Time UDP Data Emitter (RUDE) [24], lasts for 40 s. The average throughput results over the 3 experimental runs are shown in Fig. 5(b). These results demonstrate that the system is able to maintain weighted fair bandwidth sharing during the time period when the link capacity is reduced. It is also able to recover the appropriate weighted partition and utilize the entire link capacity as soon as the UDP cross-traffic flow stops. In addition, we show the measured connection throughput and advertised window for the FCS for a sample run of this experiment in Fig. 6(a) and (b) respectively.

3) Experiment 3 – Bandwidth Redistribution: This experiment is designed to show that the experimental prototype can appropriately redistribute bandwidth allocated to high-priority flows when their throughput is reduced, for example due to prolonged congestion. We send video packets from our video source, hill.cs.ucr.edu, at a rate of 496 Kbps in the form of 62 1 KB packets/s . The video client and server are the same as that used in Section VI-C2. There is a concurrent FTP session to ftp13.freebsd.org which occurs at the same time as the streaming. We assign a higher priority to the video stream and a minimum rate of 496 Kbps. From time 40 s to 100 s, the video stream reduces its rate to 160 Kbps, or 20 1 KB packets/s, to simulate external congestion affecting this particular connection. As shown in Fig. 5(c), the video stream initially receives its desired minimal rate, but during the period when it is congested, the additional bandwidth is allocated to the lower priority ftp connection. The "dip" in total bandwidth that occurs around 45 s is due to the delay required for the BWSS to determine that the high-priority flow is affected by congestion, which is not caused by the low-priority flow. When congestion subsides at 100 s, the video stream is once again allocated its desired minimal rate.

C. Streaming 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 discuss an experiment which demonstrates the benefits of streaming using the BWSS over standard TCP. Finally, we describe an experiment which offers an example of a situation in which the BWSS offers better performance than both a 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 using TCP, TCP with the BWSS, and UDP.

1) Experimental setup: All streaming experiments involve a particular video source, located in the eecs.berkeley.edu domain, and two FTP sources sending data to **Vonnegut**. At some point during each experiment, cross-traffic is sent from a different source, located in the eecs.berkeley.edu domain, to **Vonnegut**, creating congestion on the access-link. The interfering cross traffic is created using RUDE. It is important to note that the BWSS is unable to control the interfering cross-traffic in any manner.

2) Comparison of standard TCP and TCP with BWSS: The first experiment is intended to demonstrate the inadequacy of streaming over standard TCP, and the benefits for streaming applications offered by the BWSS. In this experiment, we send video packets from our video source at a rate of 496 Kbps in the form of 62 1000-byte packets/s. This is nearly equivalent to streaming video at 500 Kbps, which is now a common streaming rate used by both Windows Media Player [25] and RealOne Player [26]. There are two concurrent FTP sessions to ftp10.freebsd.org and ftp12.freebsd.org which occur at the same time as the video streaming. Furthermore, 30 s into the experiment, a 320 Kbps interfering UDP cross-traffic stream is introduced and lasts for 30 s. For this experiment, the following parameters are used for our BWSS: each FTP connection is

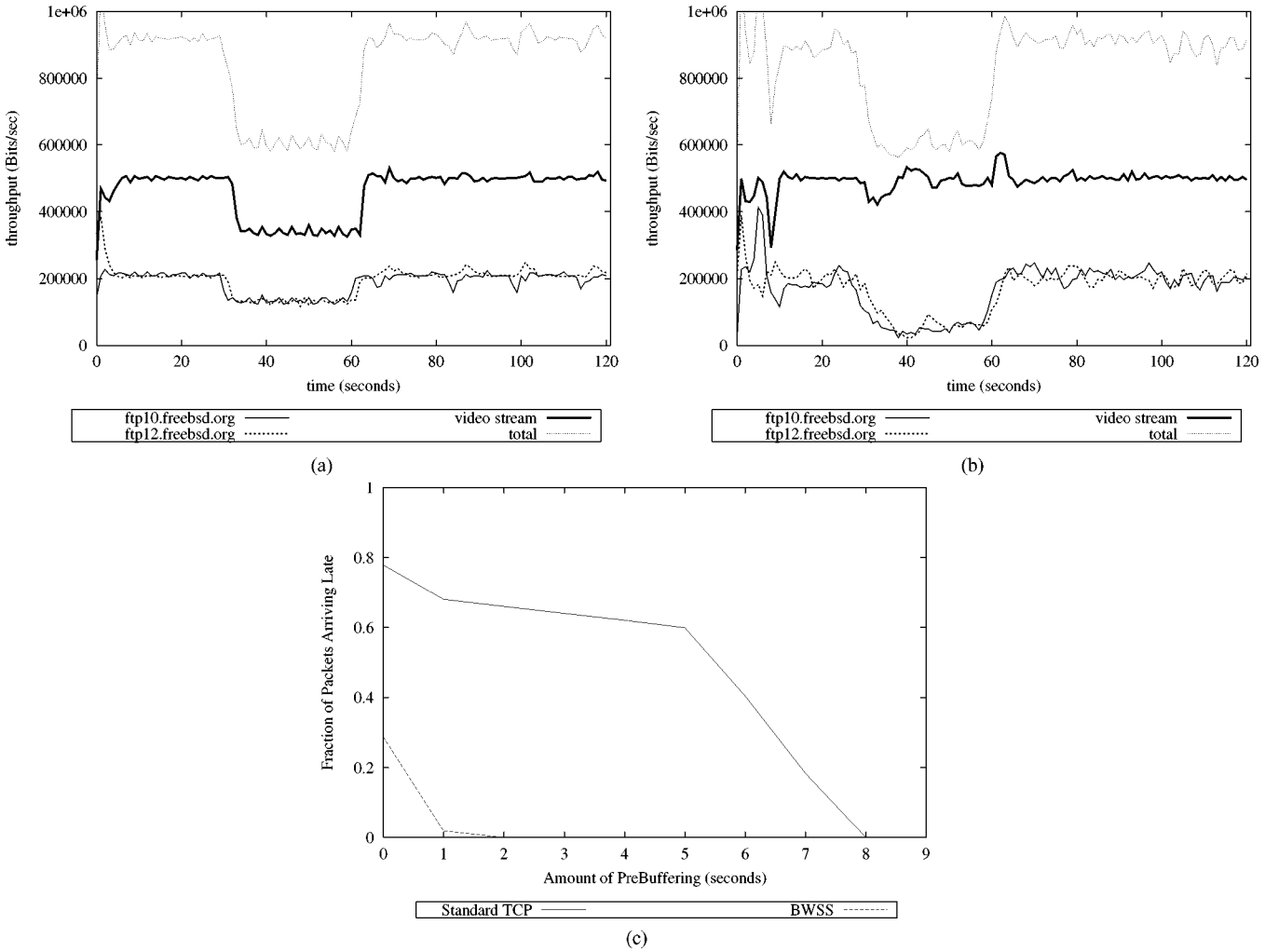

Fig. 7. Comparison of video streaming using TCP and BWSS. (a) TCP bandwidth partition, (b) BWSS bandwidth partition, and (c) fraction of late packets.

assigned no minimal rate, and a weight of 1, while the video stream is assigned a minimum rate of 496 Kbps and a weight of 0 and a higher priority than the FTP connections. This corresponds to the desire for the video to stream at 496 Kbps and to split any remaining access-link bandwidth among the FTP flows.

As shown in Fig. 7(a), while standard TCP is able to maintain the streaming rate initially, it is unable to sustain the needed rate when congestion occurs on the access-link. This would result in poorer video quality at the receiver due to frame drops, or a pause in the video to allow additional "re-buffering" of the video stream. Meanwhile, Fig. 7(b) demonstrates that the BWSS is able to maintain the desired streaming rate by appropriately reducing the rate for the FTP connections when congestion takes place. To quantify the benefits of streaming using the BWSS over standard TCP, in Fig. 7(c) we plot the fraction of packets arriving late at the receiver as a function of the number of seconds of prebuffering. Using the BWSS reduces the amount of required prebuffering by a factor of four. We note that this factor would be even larger if the congestion period had lasted longer.

3) RealVideo Streaming Experiments: We now demonstrate that streaming with TCP and the BWSS can offer better performance than 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[4] technology. SureStream technology, supported by RealNetworks's Helix Producer [27], 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: 450 Kbps, 350 Kbps, 262 Kbps, and 60 Kbps. 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 [28] at the video source used for streaming the trailer.

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

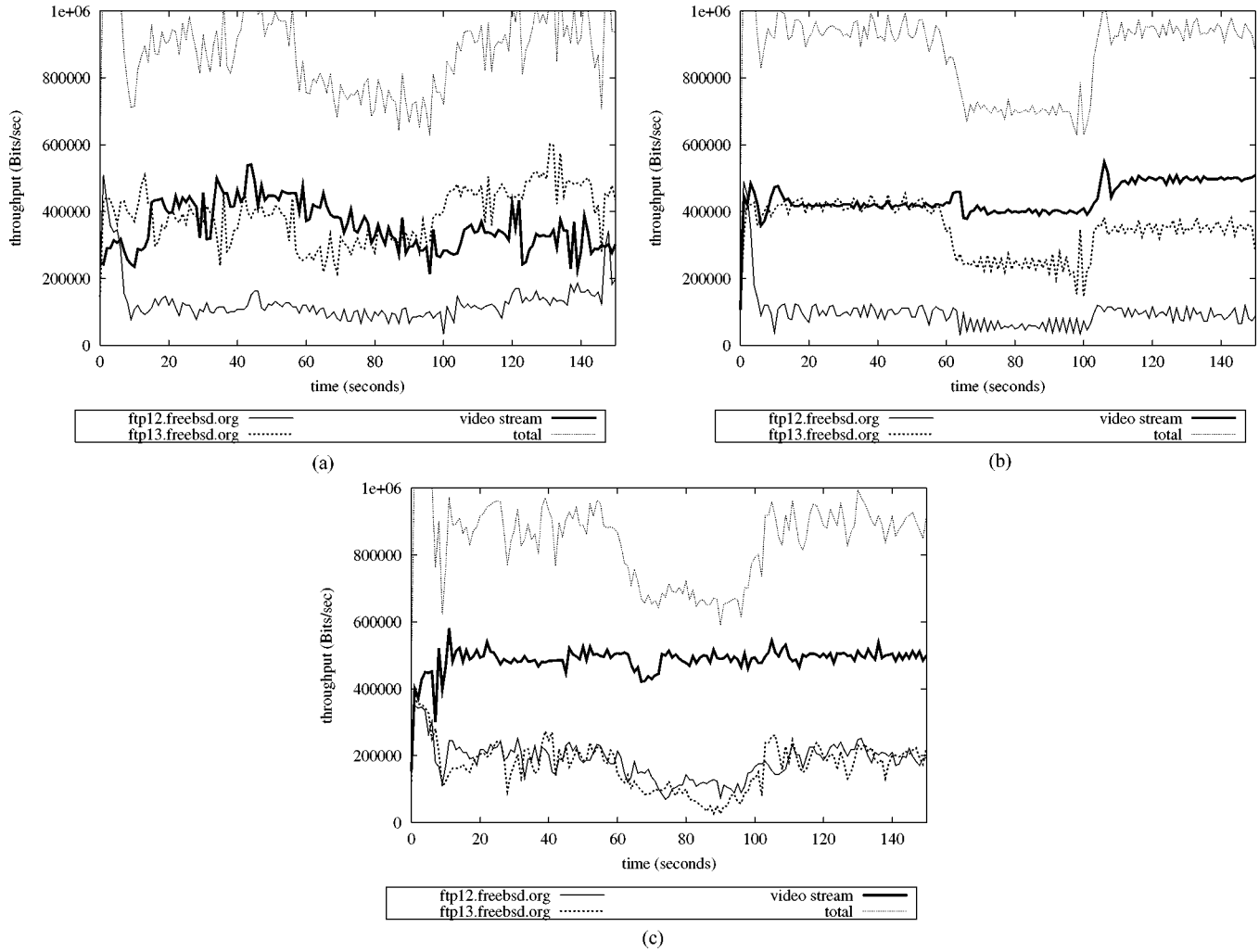

Fig. 8. Comparison of SureStream streaming using TCP, UDP and BWSS. (a) TCP bandwidth partition, (b) UDP bandwidth partition, and (c) BWSS bandwidth partition.

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 s, we introduce a 240 Kbps interfering UDP flow which lasts for 40 s. As shown in Fig. 8(a), the TCP SureStream flow is unable to maintain the necessary throughput for streaming at 450 Kbps. We observe that with TCP, the SureStream technology does not try to switch to the 350 Kbps encoding when the throughput decreases. The lack of available bandwidth causes many skipped frames and a "jerky" quality to the received video.

Meanwhile as Fig. 8(b) shows, with UDP, the SureStream technology is capable of effectively streaming at 350 Kbps[5], but does not switch to the 450 Kbps stream until the congestion has subsided at approximately 100 s into the experiment. This is evidenced by the rise in throughput for the video stream at 110 s and the switch from the 350 Kbps stream to the 450 Kbps stream is confirmed by the RealOne client. The throughput of the RealOne client using TCP with the BWSS is shown in Fig. 8(c). The video streaming connection is assigned a higher priority than the FTP applications and has a minimum rate of 520 Kbps[6] As shown in Fig. 8(c), the BWSS is able to ensure that the RealOne client has enough bandwidth to effectively stream the trailer at 450 Kbps, despite the introduction of the interfering cross-traffic. The average streaming rate for this experiment for TCP, UDP, and TCP with the BWSS are 354 Kbps, 442 Kbps, and 488 Kbps 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.

VII. Conclusions

We observe that in many cases last-hop access links are a bottleneck due to their limited bandwidth capacity. In such situations, the throughput for different TCP flows is determined by their RTT and may not coincide with the user's desires. In this work we have investigated a bandwidth-sharing system (BWSS) for TCP connections. The BWSS allows a user to specify the distribution of the access bandwidth to different flows and works

[5]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 Fig. 8(b) where the server sends the 350 Kbps encoded stream at 400 Kbps

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

without changes to the network infrastructure or sending hosts. It has been demonstrated, through real Internet experiments, that our system is able to match user preferences while achieving full utilization of the access link in many different scenarios.

Another contribution of our work is the examination of the utility of the BWSS in facilitating efficient multimedia streaming over TCP. The BWSS allows prioritization of video streaming connections by providing them with additional bandwidth when necessary. 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. Our Internet experiments 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 some challenges. For example, while TCP offers an application-independent manner of controlling the flow bandwidth by restricting the user's advertised window, UDP traffic is inherently application specific and offers no similar control knobs. Moreover, while our BWSS 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 500 Kbps 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.

In the future, 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.

ACKNOWLEDGMENT

The authors would like to thank I. Stoica for comments on an earlier draft of this paper.

REFERENCES

[1] Napster [Online]. Available: http://www.napster.com
[2] KaZaA [Online]. Available: http://www.kazaa.com
[3] Gnutella [Online]. Available: http://www.gnutella.com
[4] J. B. Postel, "Transmission Control Protocol," Information Sciences Institute, RFC 793, 1981.
[5] T. R. Henderson, E. Sahouria, S. McCanne, and R. H. Katz, "On improving the fairness of TCP congestion avoidance," in *Proc. Globecom'98*, Sydney, Australia, 1998.
[6] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair-queueing algorithm," in *Proc. ACM SigComm'89*, 1989.
[7] J. C. R. Bennett and H. Zhang, "Wf2q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM '96*, San Francisco, CA, 1996.
[8] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fiar queueing: Achieving approximately fair bandwidth allocations in high speed networks," in *Proc. ACM Sigcomm'98*, 1998.
[9] Packeteer [Online]. Available: http://www.packeteer.com
[10] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. N. Bershad, "Receiver based management of low bandwidth access links," in *Proc. INFOCOM 2000*, 2000.
[11] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing TCP," *Comput. Commun. Rev.*, vol. 28, no. 3, pp. 53–67, 1998.
[12] Y. Dong, R. Rohit, and Z. Zhang, "A practical technique to support controlled quality assurance in video streaming across the Internet," in *Packet Video*, 2002.
[13] P. Hsiao, H. T. Kung, and K. Tan, "Active delay control for TCP," in *Proc. IEEE Globecom '01*, Sep. 2001.
[14] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *Proc. ACM SIGCOMM 1999*, Sep. 1999.
[15] C. Krasic, K. Li, and J. Wapole, "The case for streaming multimedia with TCP," in *8th Int. Workshop on Interactive Distributed Mutlimedia Systems (iDMS)*, 2001.
[16] P. Hsiao, H. T. Kung, and K. Tan, "Video over TCP with receiver-based delay control," in *Proc. ACM NOSSDAV*, 2001.
[17] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly delivery of streaming media," in *Proc. IEEE ICNP '00*, Nov. 2000.
[18] S. Liang and D. Cheriton, "TCP-RTM: Using TCP for real time applications," in *Proc. IEEE ICNP '02*, 2002.
[19] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," Information Sciences Inst. , LOCATION?, RFC 1323, 1992.
[20] P. Mehra, C. D. Vleeschouwer, and A. Zakhor, "Receiver-driven bandwidth sharing for TCP," in *Proc. IEEE INFOCOM 2003*, 2003.
[21] UCB/LBNL/VINT. The Network Simulator Version 2, ns-2 [Online]. Available: http://www.isi.edu/nsnam/ns
[22] Fping [Online]. Available: http://www.fping.com
[23] NIST Net Network Emulator [Online]. Available: http://snad.ncsl.nist.gov/itg/nistnet/
[24] Real-Time UDP Data Emitter (RUDE) [Online]. Available: http://cvs.atm.tut.fi/rude/
[25] Microsoft Windows Media Player [Online]. Available: http://www.microsoft.com/windows/windowsmedia/players.asp
[26] RealOne Player [Online]. Available: http://www.real.com/
[27] Helix Producer User's Guide [Online]. Available: http://service.real.com/help/library/guides/helixproducer/Producer.htm
[28] Helix Universal Server Administration Guide [Online]. Available: http://service.real.com/help/library/guides/helixuniversalserver/realsrvr.htm
[29] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, pp. 397–413, 1993.
[30] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM 2000*, Aug. 2000.
[31] W. Tan and A. Zakhor, "Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol," *IEEE Trans. Multimedia*, vol. 1, no. 2, pp. 172–186, Jun. 1999.

Puneet Mehra received the B.S. degree in computer science from the University of California, Riverside, in 2000. From 2001 to 2003 he was a member of the Vision and Image Processing Lab in the Department of Electrical Engineering and Computer Sciences at University of California, Berkeley and received the M.S. degree in 2003. His research interests include distributed systems, networking and multimedia streaming.

Christophe De Vleeschouwer received the Electrical Engineering and Ph.D. degrees from the Université catholique de Louvain (UCL), Louvain-la-Neuve, Belgium in 1995 and 1999, respectively.

From September 1999 to November 2000, he was a Research Engineer with the IMEC Multimedia Information Compression Systems group. He is now with the Communications and Remote Sensing Laboratory of UCL, and is funded by the Belgian National Fund for Scientific Research. From August 2001 through June 2002, he was a Visiting Research Fellow at the University of California, Berkeley. His main interests concern video and image processing for communication and networking applications. He is also enthusiastic about nonlinear signal expansion techniques, and their use for signal analysis and signal interpretation.

Avideh Zakhor (F'01) received the B.S. degree from California Institute of Technology, Pasadena, and the M.S. and Ph.D. degrees from Massachusetts Institute of Technology, Cambridge, all in electrical engineering, in 1983, 1985, and 1987, respectively.

In 1988, she joined the Faculty at the University of California, Berkeley, where she is currently a Professor in the Department of Electrical Engineering and Computer Sciences. Her research interests are in the general area of image and video processing, compression, and communication. She holds five U.S. patents and is co-author (with S. Hein) of *Sigma Delta Mudulators: Nonlinear Decoding Algorithms and Stabiulity* (Norwell, MA: Kluwer, 1993).

Prof. Zakhor was a General Motors scholar from 1982 to 1983 and a Hertz Fellow from 1984 to 1988. She received the Presidential Young Investigators (PYI) Award and the Office of Naval Research (ONR) Young Investigator Award in 1992. From 1998 to 2001, she was an elected member of the IEEE Signal Processing Board of Governors. Together with her students, she has won a number of best paper awards, including from the IEEE Signal Processing Society in 1997, the IEEE Circuits and Systems Society in 1997 and 1999, the International Conference on Image Processing in 1999, and the Packet Video Workshop in 2002.