Receiver-Driven Bandwidth Sharing for TCP

Puneet Mehra and Avideh Zakhor
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
Email: {pmehra,avz}@eecs.berkeley.edu

Christophe De Vleeschouwer
Laboratoire de Télécommunications
Université catholique de Louvain, Belgium
Email: devlees@tele.ucl.ac.be

Abstract— **Applications using TCP, such as web-browsers, ftp, and various P2P programs, dominate most of the Internet traffic today. In many cases the last-hop access links are bottlenecks due to their limited bandwidth capability with users running many simultaneous network applications. Standard TCP shares bottleneck link capacity according to connection round-trip time (RTT), and may result in a bandwidth partition which does not necessarily coincide with the user's desires. We present a receiver-based control system for allocating bandwidth among TCP flows according to user preferences. Our system does not require any changes to network infrastructure, and works with standard TCP senders. NS-2 simulations, as well as actual Internet experiments, show that our system achieves desired bandwidth allocation in a wide variety of scenarios including interfering cross-traffic. We also demonstrate the viability of our system in multimedia streaming applications over 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 45Mbps, while the fastest downstream residential access is generally limited to 1.5Mbps. It is quite common for users to run multiple networking applications on a single connection, and the growing popularity of recent peer-to-peer 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.

The majority of traffic present on the Internet today is comprised of 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 coexist, 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 peer-to-peer 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.

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.

In this work we present an entirely receiver-based solution 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 and delaying TCP ACK messages to the sender. 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. The performance of the proposed systems is explored through extensive simulations using the NS-2 simulator [10] as well as through Internet experiments involving a prototype of our system for the linux operating system.

The authors of [11] 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 [12] 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 [13] 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 [14] 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 rest of the paper is organized as follows. In Section II we present an overview of our receiver-based system. In Section III we present a detailed description of the FCS for a single TCP connection. This control system is used by the BWSS discussed in Section IV. In Section V we present the simulation framework and results. In Section VI we present results from actual Internet experiments involving our prototype system. In Section VII we conclude this paper.

II. System Overview

We now provide an overview of our entirely 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 more formal definition of these goals is provided in Section IV. 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 try 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.

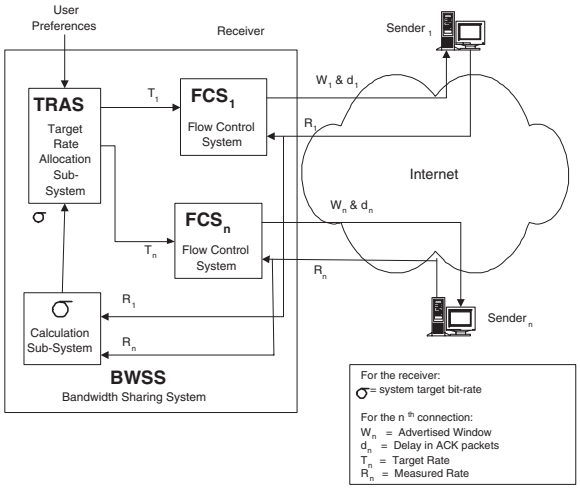

Fig. 1. Receiver-based system for bandwidth sharing

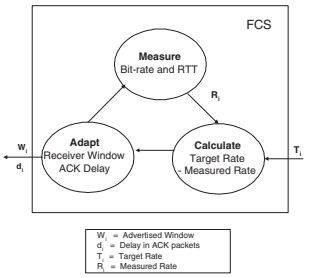

Fig. 2. TCP Flow Control System (FCS)

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 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 and delay in ACK messages to achieve the desired target bit-rate. As we describe in Section III-B, both the period of adjustment and the time scale used to measure the rate depend on the estimated round trip time and loss rate 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.

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 Fig. 1, given σ and the external user-preferences, the Target Rate Allocation Sub-system, further described in Section IV-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 Sub-system is responsible for determining the optimal value of σ which achieves full link utilization. It converges to this optimal value by iteratively increasing and decreasing the value for σ, and measuring the impact of these changes on the actual measured throughput of the different connections. This process is explained in more detail in Section IV-C. We describe each component of the BWSS, beginning with the FCS, in the following sections.

III. TCP Flow Control System (FCS)

A. System Overview

The FCS aims to maintain a particular bit-rate for a given TCP flow. 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 and the amount of delay in sending TCP ACK messages to achieve this desired target rate. The system is only effective if the desired rate is achievable under the flow's congestion window. We now present the algorithm used by the FCS to achieve the desired target bit-rate.

Our algorithm assumes an advertised window expressed as an integer number of packets. In the actual implementation, the advertised window is simply an integer multiple of the advertised TCP maximum segment size(MSS). Let w denote the advertised window, d the delay in sending ACK messages to the sender, p_{size} the size, in bits, of the packets sent, and RTT 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:

$$R = \frac{w \cdot p_{size}}{(RTT + d)} \tag{1}$$

The initialization stage follows from this model. Let T be the desired target rate for the flow. The regulation process begins by setting $d = 0$, with an initial value for w derived from (1), i.e. $w = \frac{T \cdot RTT}{p_{size}}$. 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 and delay. These refinements require a knowledge of the impact on R of a given change in w or d. The changes in R due to a change in w or d can be calculated by differentiating (1), giving:

$$\Delta R/\Delta w = p_{size}/(RTT + d) \tag{2}$$

$$\Delta R/\Delta d = -w \cdot \ p_{size}/(RTT + d)^2 \tag{3}$$

The goal of the regulation process is to bring the actual rate R of the flow within a fraction α of the target rate T, i.e.

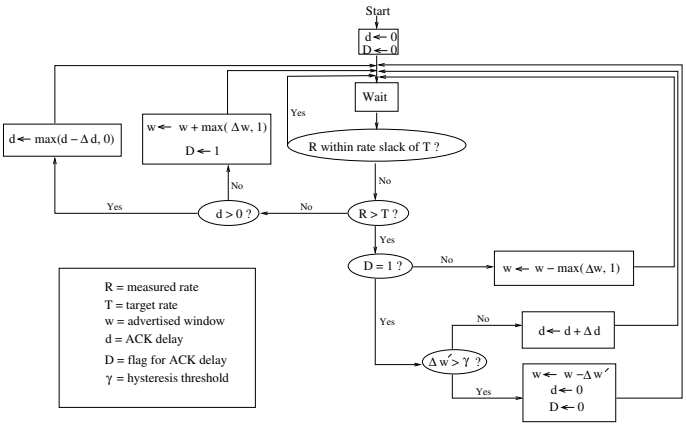

Fig. 3. TCP Flow Control System (FCS) Algorithm

$R \in [(1-\alpha)T, (1+\alpha)T]$. We will refer to α as the *rate slack*, and the corresponding interval for R as the *desired interval*. The regulation process employs two instruments to control R: the receiver's advertised window, w, and the delay in ACK messages, d. Before discussing the algorithm used by the FCS, we first enumerate the various constraints imposed on w and d. First, w is an integer and $w > 0$, which follows from the fact that we must ensure that some data is transferred during each RTT. Second, the delay d, by definition, must be non-negative. Moreover, since additional delay in ACK packets may result in unresponsiveness and instability in the TCP flow, an additional constraint for the FCS is to ensure that d is minimized, i.e. is kept as close as possible to zero.

We now outline the strategy of the FCS. Based on the need to minimize d, the simplest approach is to set $d = 0$, and only utilize w to achieve an R within the desired interval. However, since w is an integer, it is not always possible to ensure that modifying w alone will result in an R in the desired interval. Due to the fact that d is non-negative, and appears in the denominator in (1), choosing a $d > 0$ can only serve to decrease R. This, in turn, implies that given $d = 0$, the FCS should chose an integer value for w that results in an R slightly larger than T, so that d may then be increased to result in an R that is in the desired interval.

The strategy used by the regulation process is to search for the smallest w such that $R > T$. To accomplish this task, the regulation process first goes through a state in which $R < T$, and then progressively increases w until either R is in the desired interval, or $R > (1+\alpha)T$. During this window increase phase, the flag D is set to one. This is to remember the history of the system, i.e. the fact that advertising a smaller window has resulted in a rate $R < T$. In this case, if $R > (1 + \alpha)T$, the system must essentially resort to increasing d to cause R to converge to the desired interval. Decreasing w is expected to be ineffective, as reflected by the flag $D = 1$.

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 $R < T$ and $d = 0$ then based on (1), it follows

that the only way to increase R is to increase w. We increase w using (2) to determine the Δw which will lead to the desired ΔR. 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. This is to ensure that the system will converge to the smallest window for which $R > T$, without overshooting this value. Hence, incorporating the stability factor, we have

$$\Delta w = \left\lfloor \beta \cdot \frac{T - R}{\Delta R / \Delta w} \right\rfloor \quad (4)$$

Since we want to converge to the smallest window w providing a rate $R > T$, w is increased by the maximum of Δw and 1. Furthermore, we also set the flag $D = 1$ in case the increase in w results in $R > (1 + \alpha)T$. Since R is less than T before the increase in w, if we find that $R > (1 + \alpha)T$ after the increase, then the system must increase d in order to cause convergence to the desired interval. The flag $D = 1$ is used by the system to distinguish the need to increase d from the more desirable approach of decreasing w.

3) If $R < T$ and $d > 0$ then we are in situation where we had previously increased d to converge to the desired interval, but now find that $R < (1 - \alpha)T$. Since one of the constraints is to minimize d, we first attempt to reduce d before considering any modifications to w. Specifically, we reduce the amount of delay in ACK messages using (3) to determine the necessary Δd.
4) If $R > T$ and $D = 0$ then the previous system state was such that $R > T$. At this point, according to (1), the system may reduce R by either increasing d or decreasing w. Since we have not ascertained that an increase in d is essential to converge to the desired interval, w is decreased, using (2) to compute the necessary Δw. This process continues until either T is achieved or the rate R falls below the target T.
5) If $R > T$ and $D = 1$, then the system is expected to have converged to the smallest window for which $R > T$, and it enters the *delay increase* stage. During simulations, our system rarely ran into this step, which means that adjusting the window is often sufficient to reach the desired interval. However we describe this step in detail both for completeness, and because this might become important in case of coarse window adjustment granularity, for example due to a small RTT. As outlined above, the flag $D = 1$ is set when $R < T$ and w is being slowly increased to converge to the desired interval. Since R is now above T, we expect that the system has achieved the smallest window such that $R > T$. Hence it is desirable to increase d, in accordance with (3), to cause R to converge to the desired interval.

   Increasing d is effective the first time, or perhaps the first few times, the system enters this state. However, values of R are continually being updated in the background and as such, the conditions that drove the system to this state might be obsolete after a while, due to the dynamic nature of network conditions. This implies that adjusting w could still be an option in this state. Hence our approach is to devise a conditional solution in which w is adjusted only if the required change $\Delta w'$, is larger than a threshold, γ; otherwise d is adjusted. This solution is shaped by our desire to minimize d. Specifically, the system calculates the window decrement $\Delta w'$ which would be required to achieve T, assuming the delay is reset to zero. In practice, $\Delta w'$ is computed as follows:

$$\Delta w' = \left\lfloor \frac{R - T}{\Delta R / \Delta w} + \frac{-d \cdot \Delta R / \Delta d}{\Delta R / \Delta w} + 0.5 \right\rfloor \quad (5)$$

   The first term of the sum compensates for the mismatch between T and R, while the second corresponds to the increase of rate expected from the delay reset to zero. As long as $\Delta w' < \gamma$, the system stays in the delay increase stage and continues to increase d to achieve T. The *hysteresis threshold* γ is designed to prevent small differences between R and T from causing oscillations in w. In practice, in our simulations and experiments, $\gamma = 2$.

   If the $\Delta w'$ computed based on (5) is larger than γ, i.e. $\Delta w' > \gamma$, then the system exits the delay increase stage and decreases w to achieve T.

After any change in the advertised window or ACK delay, 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 good 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 [15]. 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 rate regulation system. Our system estimates R at the end of successive bandwidth-estimation periods. The procedure relies on an exponentially-weighted moving average, detailed in the Appendix. Due to the iterative nature of the regulation system and to the loss-related TCP throughput fluctations, the bandwidth-estimation period ϕ offers a tradeoff between the accuracy of bandwidth measurement and the time needed to converge to the target rate. The value of ϕ is defined accurately in the Appendix as a function of the RTT, the window size, and the timescale over which the flow experiences loss-related throughput fluctuations.

The last major characteristic of the FCS that needs to be discussed is the frequency of adjustment of the window and delay parameters. 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, $\varphi > RTT + \phi$. In practice, since we have not synchronized the start of bandwidth measurement with the beginning of an adjustment, we set $\varphi > RTT + 3 \cdot \phi$ to ensure enough time for the adjustment to have an effect on throughput.

IV. Bandwidth Sharing System (BWSS)

In Section III, 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 IV-C, 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 system due to space limitations, 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 pre-coded 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 and the delay in ACK messages as the *target bit-rate* for a flow, or simply the *target rate*. As mentioned in Section II, 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 Sub-system, discussed later in Section IV-C, 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 $\{T_i\}_{0 \le i < N}$ 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:

$$\sum_{i=0}^{N-1} T_i = \sigma. \tag{6}$$

Note that there are many $\{T_i\}$ which satisfy (6), 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 a 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. Let T_i and σ be defined as noted above. The definition of a 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 $\sigma \ge \sum_{j=0}^{N-1} m_j$. In this case, any *remaining* bandwidth should be shared among the flows according to their respective weights. More formally, in this case, $\{T_i\}_{0 \le i < N}$ is a desired partition if $\forall i$

$$T_i = m_i + w_i \cdot \frac{\sigma - \sum_{j=0}^{N-1} m_j}{\sum_{j=0}^{N-1} w_j} \tag{7}$$

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 decreasing priority, thus $p_i > p_j, \forall i < j$. Then $\{T_i\}_{0 \leq i < N}$ is a desired partition if $\forall i$

$$T_i = min(m_i, max(0, \sigma - \sum_{j=0}^{i-1} m_j)) \tag{8}$$

The formulation in (8) states that in a 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.

B. System Overview and Terminology

Before we describe the BWSS in more detail, we introduce some terminology. A *receiver flow* is a flow whose destination is the receiver running the BWSS. An *external flow* is a flow destined to another receiver.

The philosophy behind the BWSS is quite simple: basically, the BWSS *probes* the receiver flows, i.e. it increases or decreases a flow's target bit-rate by adjusting σ, in order to iteratively converge to an optimal bandwidth 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.

We refer to flows which have their ideal throughput limited below the TCP rate due to the FCS as being *constrained*. The BWSS initialization estimates the total available receiver bandwidth by releasing the constraints on all the flows, and sets σ to this value. Given σ, the Target Rate Allocation Sub-system (TRAS) defines the target rates for all the flows. After the initialization stage, the system iterates between two different stages. During the first stage, the system increases σ, thereby increasing the target bit-rate for all of the flows. We refer to this stage as the *relaxing* phase, since the BWSS tries to relax the throughput constraints of constrained flows in the system. The goal of this stage is to better utilize the link capacity subject to the constraint that user preferences are met. In contrast, the second stage, or *constraining* phase, serves to further limit the throughput of the constrained flow(s). The purpose of this new constraint is to make some additional bandwidth available in the hope that it benefits a flow that is more *bandwidth deserving* than the constrained flow(s), without affecting the global utilization. The condition for a flow i being more deserving than a flow j depends on the receiver preferences, and on the actual rates of both flows. Let R_i and R_j be the actual rates of the i^{th} and j^{th} flow respectively. If $R_i < m_i$, flow i is more deserving than j if $p_i > p_j$, since a higher priority flow should have its minimal requirements met before a lower priority one. If $R_i \geq m_i$, i is more deserving than j if $R_j \geq m_j$ and $R_i/w_i < R_j/w_j$, which captures the notion that i has not received its weighted share of bandwidth.

C. σ Calculation Sub-System

Before delving into the details of the σ Calculation Sub-system, 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 Sub-system(TRAS). These T_i are input into the appropriate FCS for the different flows, which compute the delay and 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 $U = \sum_i R_i$. The σ Calculation Sub-system waits for the changes made by the FCS to take effect and then computes U to decide whether to increase or decrease σ. Since the T_i are computed by the TRAS, they always match user preferences. Thus the main goal of the σ Calculation Sub-system is to choose σ to fully utilize the link capacity.

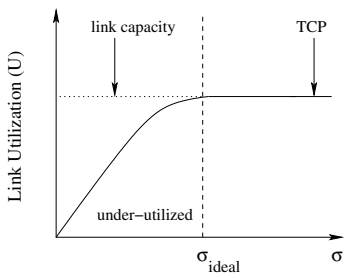

Fig. 4. Link Utilization as a function of σ

Fig. 4 shows the dependency between σ and the resulting values of link utilization, U. 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 optimal set of actual rates, U_{ideal}, which 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 Sub-system. The goal is to find σ_{ideal},

the smallest σ, which achieves full link utilization, and the corresponding desired partition.

The principle of the BWSS is simple, and based on the two stage probing process previously described. Probing in the system consists of either increasing or decreasing σ, and calculating the desired partition, and hence the target bit-rates for the different flows in the system, corresponding to the given value of σ. Note that for flows which are limited by a network bottleneck, i.e. for which the sender window is bounded by the congestion window rather than by the receiver advertised window, R_i may be less than T_i, and hence there may be cases when $U < \sigma$. For this reason, an increase of σ is an attempt at improved link utilization if it is not already fully utilized. Link utilization is improved if there is an increase in the aggregate actual rates as measured by the system, i.e. an increase in U, due to the increase in σ. On the other hand, a decrease of σ is an attempt to better fit the receiver preferences. To better understand this, it helps to view standard TCP as a special case of our system in which the receiver's advertised window does not constrain the flow, and hence $\forall i,\ T_i = \infty$ and $\sigma = \infty$. If the target bit-rates for the flows in the system are not below their standard TCP rate, then our proposed system degenerates to a classical TCP bandwidth partition. At the other extreme, a value of σ close to zero results in the starvation of most flows and only allocates bandwidth to the most deserving flows.

There is an initialization phase which is done at system startup, in response to a change in user preferences, and upon opening or closing a connection. During this stage the BWSS estimates the total available receiver bandwidth by releasing the throughput constraints on all flows, and sets the initial σ to this value. Since some of the flows might be limited by network bottlenecks, this initial partition does not necessarily provide full link utilization. However, subsequent iterations of the BWSS ensure convergence toward full link utilization while matching user preferences for the bandwidth partition.

The main difference between the actual implementation and the conceptual description presented above is that the σ Calculation sub-system in the actual implementation aims to increase σ by probing a single flow at a time. This is to handle the situation when minimal bandwidth requirements cannot be met for all flows. In this situation a small change to σ by the system only affects the target rate of a single flow. If this flow is limited by a network bottleneck, releasing it will not increase U. To efficiently use any available link capacity, the flows are probed independently and in decreasing order of priority. Each probing event increases the target rate of a single flow and checks for an increase in U. If there is no increase then the target rate is reset to its initial value. Otherwise the target rates of other flows are adapted to the desired partition based on the probed flow's new target rate. Since this adaptation only increases the target rate of flows which have previously been unsuccessfully probed, it does not have any impact on the system. If σ is larger than the sum of the minimal rates, then individual probing is not necessary.

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 only constraints the sender's congestion window.

V. Simulation Results

We have tested the BWSS, using the NS-2 simulator, under a variety of different scenarios to illustrate the salient features of the system. We have tested the system using routers employing droptail FIFO buffering as well as RED gateways [16]. The results are presented for the simulations using RED gateways with a buffer of 50 packets and a minimum threshold of 30 packets and a maximum threshold of 40 packets. The default values present in version 2.1b8 of NS-2, were used for the rest of the buffer parameters. Each scenario has been run multiple times with slight variations in the starting time of interfering flows and the time when the system adjustment process was started. We use the measured throughput of different flows as the performance criteria for our proposed system.

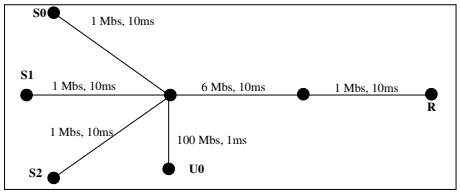

Fig. 5. Topology for Scenario 1 - Ideal Case

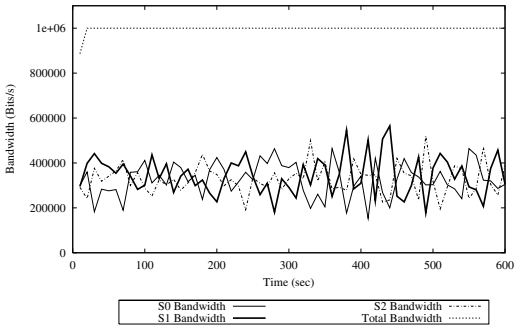

Fig. 6. TCP Bandwidth partition for Scenario 1 - Ideal Case

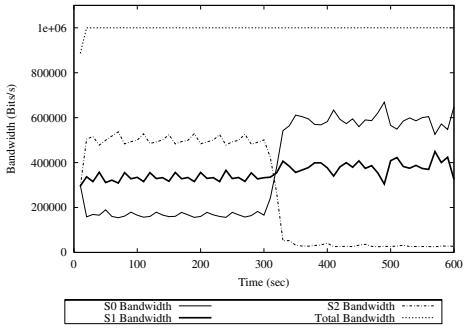

Fig. 7. System Bandwidth partition for Scenario 1 - Ideal Case

A. Scenario 1 - The Ideal Case

The topology for the simplest scenario tested is shown in Fig. 5. It consists of 3 ftp senders S0 through S2 which are bandwidth limited due to the 1Mb/s bottleneck link at the receiver R. The UDP sender, U0 depicted in Fig. 5, is not

used in this example. The bandwidth partition due to standard TCP is shown in Fig. 6. Since all of the flows have the same RTT to the receiver, they all receive an equal share of the bandwidth. For our system, the ftp flows are assigned increasing weights of 1,2, and 3 for senders S0, S1, and S2, but decreasing priority. Thus sender S0 has the highest priority, but the lowest weight. Initially the minimum rate is zero for all flows, and bandwidth is shared according to flow weights. At time 300, user preference specifies a minimum rate of 600Kb/s for all flows, which cannot be achieved, and hence the system satisfies the minimum requirements according to flow priority, with S0's flow receiving approximately 600Kb/s, and S1 receiving most of the remaining 400Kb/s. The bandwidth partition of our control system, shown in Fig. 7, matches our expectations of how it should work.

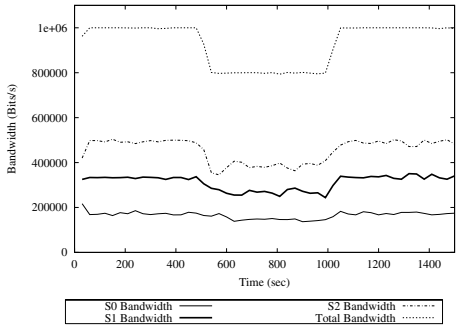

Fig. 8. Bandwidth partition for Scenario 2 - link bandwidth reduction

B. Scenario 2 - Link Bandwidth Reduction

The following scenario investigates the ability of the system 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. The topology used is the same as shown in Fig. 5. There is no minimum bandwidth requirement for any of the flows, and the weights assigned to the flows are 1,2, and 3 for senders S0, S1, and S2, respectively. A 200Kb/s UDP flow originating at node U0 and terminating at the receiver R is started at 500 seconds and stopped at 1000 seconds into the simulation. As shown in Fig. 8, the system is able to maintain the weighted fair bandwidth sharing during the time period when the link capacity is effectively reduced to 800Kb/s by the interfering UDP flow. It is also able to recover the appropriate weighted partition and utilize the entire link capacity as soon as the UDP flow stops.

C. Scenario 3 - TCP Congestion

We now explore the system's ability to react to congestion affecting a particular flow. Specifically, this scenario demonstrates that the system is able to redistribute bandwidth allocated to high-priority flows when they experience long periods of congestion. The topology used in this scenario is depicted in Fig. 9. The simulation consists of the 3 ftp senders S0 through S2 which are ordered in increasing order of priority and have a minimum bandwidth requirement of 600Kb/s, 100Kb/s and 600Kb/s respectively. There are 20 interfering ftp flows, I0 through I19, which are limited to 35Kb/s by their link capacity, traversing a link used by sender S2. Thus the aggregate amount of cross-traffic is approximately 700Kb/s. The interfering flows begin at time 400s and end at time 1200s.

As shown in Fig. 10, the traffic from sender S2 is initially able to get its desired 600Kb/s minimum rate until the TCP cross-traffic is started, at which point it is limited to 300Kb/s due to congestion on the shared link. The system is able to determine that the flow is limited by a network bottleneck and hence assigns additional bandwidth to the lower priority traffic from S0, which has not achieved its minimum requirement of 600Kb/s, to better utilize the receiver's link capacity. When the TCP cross-traffic is stopped at 1200 seconds, the system is able to determine that the flow from S2 is no longer constrained by any bottlenecks in the network, and is able to redistribute a larger share of the link capacity to the flow.

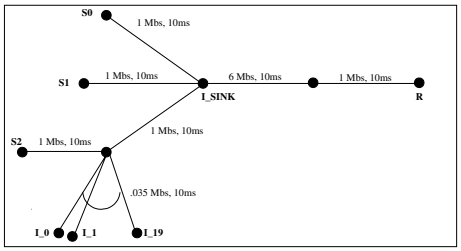

Fig. 9. Topology for Scenario 3 - TCP Congestion

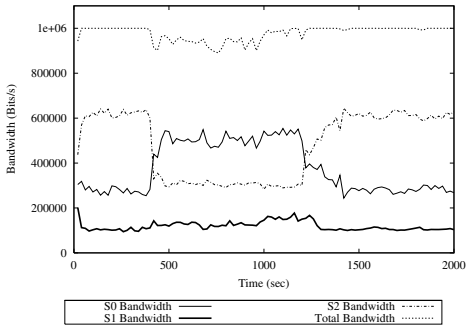

Fig. 10. Bandwidth partition for Scenario 3 - TCP Congestion

D. Scenario 4 - Large Differences in RTT

The next scenario is designed to highlight TCP's bias against flows with large RTT, and our system's ability to counter this inherent unfairness and to prevent the starvation of flows with large RTT. The topology used for this scenario is shown in Fig. 11. Senders S0,S1, and S2 have an RTT of 30ms, 120ms and 300ms, respectively. For the first 500 seconds of the simulation we do not turn on the control system and allow standard TCP operation. As shown in Fig. 12, since S0 has the smallest RTT, it obtains the largest share of the link bandwidth. At 500 seconds we start the system and specify a minimum rate of 600Kb/s for each flow, with increasing priority among the senders S0 through S2. The ftp connection from S2 is able to achieve the desired rate of 600Kb/s and most of the remaining bandwidth is distributed to sender S1 which has the next highest priority.

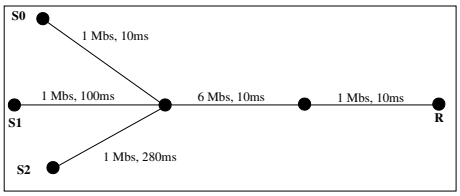

Fig. 11. Topology for Scenario 4 - Large Differences in RTT

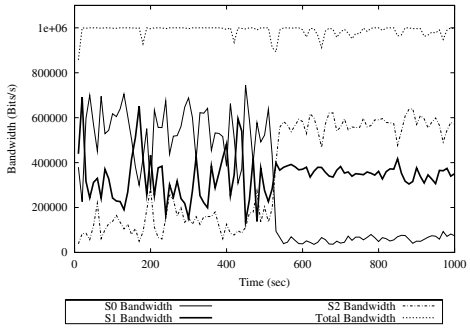

Fig. 12. Bandwidth partition for Scenario 4 - large differences in RTT

E. Scenario 5 - Multimedia Streaming

In this section we motivate another application of our system, namely multimedia streaming over TCP. Traditionally most streaming applications have been implemented using UDP coupled with a rate control protocol which ensures fairness with TCP traffic [17]. However, there have been several recent proposals, [14], [18], and [19], which challenge the conventional wisdom that TCP is unsuitable for multimedia streaming. Our proposed system shares this goal of providing effective video streaming over TCP. The topology for this example is shown in Fig. 5. We assume that sender S1 wishes to stream a video at 450Kb/s to the receiver, while S0 has ftp data traffic. Source S2 is not used in this scenario. We generate 300Kb/s of interfering UDP cross-traffic from sender U0 to R beginning at time 400s and lasting until 1000s into the simulation. To achieve better performance than TCP using our system, we assign a minimum rate of 450Kb/s, to the traffic from sender S1, while the rest of the traffic is dedicated to the ftp flow by assigning a minimum rate of ∞, so that it uses the remaining link capacity. As shown in Fig. 13, our system aims to maintain the minimal rate for the video application at the expense of the ftp traffic when the link bandwidth is reduced. To measure the benefits in streaming video, we recorded the number of packets which would have been past their deadline, as a function of the amount of pre-buffering of video done at the receiver, given the assumption that the receiver has unlimited space to prebuffer video. The results are shown in Fig. 14. As shown, the transient bandwidth decrease of the conventional TCP requires significant prebuffering. Without any streaming-specific modifications to our system, we are able to achieve a factor of 6 reduction in the amount of prebuffering that must be done to avoid any late packets. These results indicate the potential of our system for effective multimedia streaming over TCP. In the future, we intend to trade-off packet losses for shorter delays by acknowledging lost packets while controlling the advertised window to maintain a TCP-friendly rate.

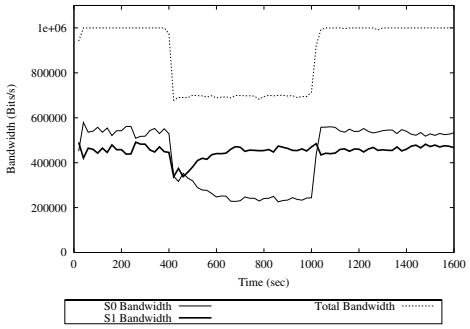

Fig. 13. System Bandwidth partition for Scenario 5 - multimedia streaming

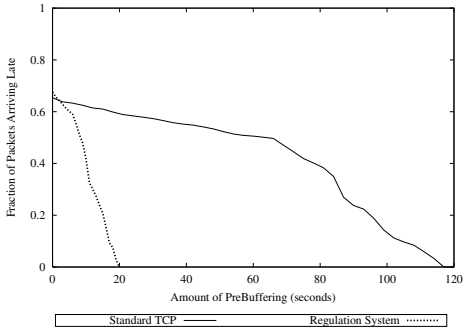

Fig. 14. Fraction of packets arriving Late

VI. Internet Experiments

We have implemented an initial prototype of our proposed flow control system (FCS) and bandwidth sharing system (BWSS) for the linux operating system. We highlight some relevant implementation details and then present preliminary results from Internet experiments.

A. 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 pre-load 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 delay between ACK messages is not used in the current prototype). 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 using the fping [20] program, which relies on Internet Control Message Protocol (ICMP) echo requests.

Due to the constraints of our user-space implementations, our current prototype does not keep track of lost packets and hence the bandwidth estimation period for a given connection is simply based upon its TCP RTT. Furthermore, since it is not possible to control the delay in ACK packets from user-space, our prototype only modifies the receiver's advertised window.

B. Experimental Methodology and Results

We now present the methodology and results of two experiments conducted with actual Internet hosts which duplicate the first two simulations in Section V. Each experiment consists of FTP downloads of FreeBSD ISO images from the different ftp servers. The receiving host running the BWSS is a PC workstation running Mandrake Linux 8.1 with kernel version 2.4.8-26. The receiving host is connected to the Internet with a cable-modem connection provided by AT&T Broadband in Berkeley, California. The BWSS is started after 30 seconds of initial TCP operation in both experiments. Each experiment was also conducted multiple times and the average over these runs is graphed. The throughput data is collected using the tcpdump utility [21]. Data for each specific connection is then parsed using tcptrace [22].

C. Experiment 1 - Scenario 1

This experiment duplicates Scenario 1 from Section V. We assign weights of 1,2, and 3 to hosts ftp13.freebsd.org, ftp12.freebsd.org, and ftp14.freebsd.org respectively. We assign priorities in the reverse order, namely 3,2, and 1 for the host order listed above. Initially we do not set any minimal rate, and thus the bandwidth is distributed according to weight. At time 90, we assign a minimal rate of 100Kbytes/s for each flow and restart the BWSS. As shown in Fig15, the prototype achieves the expected flow throughput.

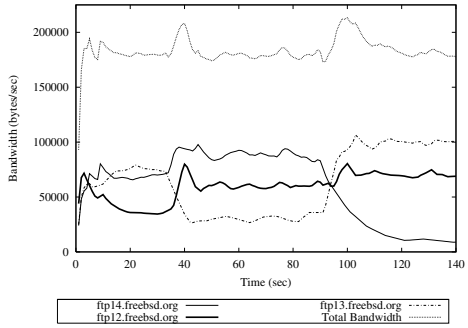

Fig. 15. System Bandwidth partition for Internet Experiment 1

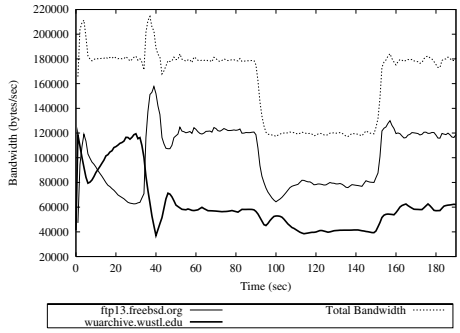

Fig. 16. System Bandwidth partition for Internet Experiment 2

D. Experiment 2 - Scenario 2

This experiment duplicates Scenario 2 from Section V. We assign a weight of 2 to the connection from ftp13.freebsd.org and a weight of 1 to the download from wuarchive.wustl.edu. No minimal rate is set, so the flow priority is not relevant. At time 90, a 60Kbyte/s UDP stream, terminating at the receiver host, is started from a host located in the eecs.berkeley.edu domain. The UDP traffic lasts for 60 seconds. The UDP stream is generated using the Real-Time UDP Data Emitter (RUDE) [23]. Once again, the results from this experiment shown in Fig 16 confirm the behavior observed during simulation.

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 presented our preliminary investigation of a bandwidth-sharing system for TCP connections that allows a user to specify the distribution of the access bandwidth to different flows. It has been demonstrated, through simulations and real Internet experiments, that our system is able to match user preferences while achieving full utilization of the receiver's access in many different scenarios. In future work we intend to examine the utility of our system in facilitating efficient multimedia streaming over TCP.

Acknowledgments

This work was supported by NSF under grant CCR-9979442 and AFOSR contract F49620-00-1-0327. Christophe

De Vleeschouwer was a visitor at UC Berkeley, with support from Belgian NSF, while this work was being completed. The authors would like to thank Ion Stoica for comments on an earlier draft of this paper and the anonymous reviewers who helped improve the quality of this paper.

APPENDIX

Our system estimates R at the end of successive bandwidth-estimation periods. This Appendix describes how the rate estimation is performed, and how the bandwidth-estimation period is chosen.

Let R_ϕ be the number of bytes received per unit of time over the bandwidth-estimation period ϕ. Our system uses an exponentially-weighted moving average to adjust R. Let ϵ be the exponential average parameter; the rate estimation is adjusted as follows: $R \leftarrow \epsilon \cdot R + (1-\epsilon)R_\phi$. In our simulations ϵ has been set to 0.3. 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 value of ϕ offers a tradeoff between the accuracy of bandwidth measurement and the time needed to converge to the target rate. Specifically, in our system, the value of ϕ depends on the timescale over which the flow experiences loss-related throughput fluctuations. Whenever a TCP flow experiences a loss, the sender halves its congestion window and enters the congestion avoidance phase, in which the window is increased by 1 packet during each RTT period. Thus, after a loss, a flow will need approximately $\frac{1}{2}w \cdot RTT$ time to recover its throughput.

Let I be the time between losses and let w be the advertised window. The case $I > RTT \cdot w/2$ corresponds to a stable TCP throughput, which appears when the congestion window is effectively constrained by the advertised window. In this situation, a local measurement of the bandwidth is likely to provide a good estimation of the average throughput. So, if $I > RTT \cdot w/2$, we choose $\phi = 2 \cdot RTT$. Otherwise, in the case of frequent losses, the period ϕ required to obtain an accurate bandwidth estimate should be proportional to the time interval I between consecutive losses. We aim to compromise between estimation accuracy and time for convergence and choose $\phi = \frac{2}{5} \cdot I$ when $I < RTT \cdot w/2$.

In order to estimate the time between losses I, we adopt techniques used in TFRC [24]. In TFRC, the notion of a *loss-event* groups all losses incurred during a short time period into a single *event*. The interval I is then estimated based on a weighted average of the *loss intervals*, which are the time intervals measured between a number of consecutive recent loss-events[24]. The technique proposed in [24] weighs the last 8 loss intervals to compute an average. This technique is slow to adapt to improved network conditions. Hence our proposed system makes a slight modification to the scheme proposed in [24]. Let $ctime$ be the time since the last loss event. Let I' be the weighted average of past loss-intervals, as computed in [24]. Then our system computes the average time between losses, I, as follows:

$$I = (1 - e^{-ctime/I'}) \cdot ctime + e^{-ctime/I'} \cdot I'$$

The reasoning behind the exponential weighting factor is that if the time since the last loss event is sufficiently larger than the weighted average of past intervals, then the overall average should be more influenced by this measurement. As $ctime \rightarrow \infty$ then $I \rightarrow ctime$. This approach allows a faster adaptation to improved network conditions, i.e. when a loss has not been observed for a long time after a period of congestion.

REFERENCES

[1] "Napster. http://www.napster.com," .
[2] "KaZaA. http://www.kazaa.com," .
[3] "Gnutella. http://www.gnutella.com," .
[4] J.B. Postel, "Transmission Control Protocol," RFC 793, Information Sciences Institute, September 1981.
[5] Thomas R. Henderson, Emile Sahouria, Steven McCanne and Randy H. Katz, "On Improving the Fairness of TCP Congestion Avoidance," in *Proceedings of Globecom'98*, Sydney, AU, 1998.
[6] A. Demers, S. Keshav and S. Shenker, "Analysis and Simulation of a Fair-queueing algorithm," in *Proceedings of ACM SigComm'89*, 1989.
[7] Jon C.R. Bennett and Hui Zhang, "Wf2q: Worst-case Fair Weighted Fair Queueing," in *Proceedings of IEEE INFOCOM '96*, San Francisco, CA, 1996.
[8] Ion Stoica, Scott Shenker and Hui Zhang, "Core-Stateless Fiar Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proceedings of ACM Sigcomm'98*, 1998.
[9] "Packeteer. http://www.packeteer.com," .
[10] "UCB/LBNL/VINT. The Network Simulator version 2, ns-2. http://www.isi.edu/nsnam/ns," .
[11] N.T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson and B.N. Bershad, "Receiver Based Management of Low Bandwidth Access Links," in *Proceedings of INFOCOM 2000*.
[12] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing tcp," 1998.
[13] 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.
[14] Pai-Hsiang Hsiao, H.T. Kung and Koan-Sin Tan, "Active Delay Control for TCP," in *Proceedings of IEEE Globecom '01*, September 2001.
[15] Van Jacobson, R. Braden and D. Borman, "TCP Extensions for High Performance," RFC 1323, Information Sciences Institute, May 1992.
[16] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.
[17] 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.
[18] B. Mukherjee and T. Brecht, "Time-Lined TCP for the TCP-Friendly Delivery of Streaming Media," in *Proceedings of IEEE ICNP '00*, November 2000.
[19] Pai-Hsiang Hsiao, H.T. Kung and Koan-Sin Tan, "Video over TCP with Receiver-based Delay Control," in *Proceedings of ACM NOSSDAV*, 2001.
[20] "Fping. http://www.fping.com," .
[21] "tcpdump. http://www.tcpdump.org," .
[22] "tcptrace. http://www.tcptrace.org," .
[23] "Real-Time UDP Data Emitter (RUDE) http://cvs.atm.tut.fi/rude/," .
[24] Sally Floyd, Mark Handley, Jitendra Padhye and Jorg Widmer, "Equation-Based Congestoin Control for Unicast Applications," in *Proceedings of ACM SIGCOMM 2000*, August 2000.
[25] D. Katabi, I. Bazzi, and X. Yang, "A passive approach for detecting shared bottlenecks," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*, Arizona, October 2001.
[26] D. Rubenstein, J.F. Kurose, and D.F. Towsley, "Detecting shared congestion of flows via end-to-end measurement," in *ACM SIGMETRICS: Measurement and Modeling of Computer Systems*, Santa Clara, CA, June 2000, pp. 145–155.