2003-12-02

Providing Soft Bandwidth Guarantees Using Elastic TCP-based Tunnels

Guirguis, Mina

Boston University Computer Science Department

Providing Soft Bandwidth Guarantees Using Elastic TCP-based Tunnels [*]

MINA GUIRGUIS
msg@cs.bu.edu

AZER BESTAVROS
best@cs.bu.edu

IBRAHIM MATTA
matta@cs.bu.edu

NIKY RIGA
inki@cs.bu.edu

GALI DIAMANT
gali@cs.bu.edu

YUTING ZHANG
danazh@cs.bu.edu

Computer Science Department
Boston University
Boston, MA 02215, USA

BUCS-TR-2003-028

Abstract

The best-effort nature of the Internet poses a significant obstacle to the deployment of many applications that require guaranteed bandwidth. In this paper, we present a novel approach that enables two edge/border routers—which we call Internet Traffic Managers (ITM)—to use an adaptive number of TCP connections to set up a tunnel of desirable bandwidth between them. The number of TCP connections that comprise this tunnel is elastic in the sense that it increases/decreases in tandem with competing cross traffic to maintain a target bandwidth. An origin ITM would then schedule incoming packets from an application requiring guaranteed bandwidth over that elastic tunnel. Unlike many proposed solutions that aim to deliver soft QoS guarantees, our elastic-tunnel approach does *not* require any support from core routers (as with IntServ and DiffServ); it is scalable in the sense that core routers do not have to maintain per-flow state (as with IntServ); and it is readily deployable within a single ISP or across multiple ISPs. To evaluate our approach, we develop a flow-level control-theoretic model to study the transient behavior of established elastic TCP-based tunnels. The model captures the effect of cross-traffic connections on our bandwidth allocation policies. Through extensive simulations, we confirm the effectiveness of our approach in providing soft bandwidth guarantees. We also outline our kernel-level ITM prototype implementation.

Keywords: TCP, Congestion Control, Control Theory, Transient Performance, Simulation.

1. Introduction

The scalability of the Internet hinges on our ability to tame the unpredictability associated with its open architecture. Significant and unpredictable changes in network dynamics (and hence performance) make it harder on applications to adequately perform and even adapt if they are designed to do so. To that end, significant efforts have been expended in order to extend the basic best-effort Internet Protocol (IP) architecture so it provides hard or soft performance guarantees (on bandwidth, delay, loss, etc.) Such performance guarantees are needed by applications sensitive to Quality-of-Service (QoS), *e.g.* real-time, video streaming and games.

The IntServ architecture [2] extends IP to provide hard performance guarantees to data flows by requiring the participation of *every* router in a per-flow resource allocation protocol. The need to keep per-flow state at every router presents significant scalability problems, which makes it quite expensive to implement. To that end, the DiffServ architecture [15] provides a solution that lies between the simple but QoS-oblivious IP, and the QoS-aware but expensive IntServ solution. DiffServ encompasses the scalable philosophy of IP [14] in pushing more functionality toward the edges leaving the core of the network as simple as possible. Nevertheless, DiffServ has not yet been successful in being widely deployed by Internet Service Providers (ISPs). One reason is that DiffServ solutions still require some support from core routers (albeit much less than that of IntServ solutions). For example, the DiffServ solution proposed in [7] requires the use and administration of a dual (weighted) Random Early Drop (RED) queue management in core routers.

In addition to the need of IntServ-based and DiffServ-based solutions for network/router support, such solutions typically assume that *all* flows going through the network are managed.[1] For example, with both IntServ and DiffServ,

[*] This work was supported in part by NSF grants ANI-0095988, ANI-9986397, EIA-0202067 and ITR ANI-0205294, and by grants from Sprint Labs and Motorola Labs.

[1] For instance, typical DiffServ solutions assume that all edge routers perform necessary admission control and packet classification.

there are no provisions for ensuring fairness amongst unmanaged best-effort flows to effectively use *excess* bandwidth in the network. We believe this to be a main drawback of these approaches as they do not lend themselves to incremental deployment on a wide-scale.

Guaranteed Throughput over Best-Effort Networks: In this paper, we investigate a solution that enables the delivery of *soft bandwidth guarantees* through the use of a best-effort, QoS-oblivious networking infrastructure. Unlike both IntServ and DiffServ, our approach does not require *any* modifications to core routers and is designed in such a way so as it may co-exist with best-effort traffic.

Our approach to delivering soft bandwidth guarantees between two points is to adaptively adjust the demand from the underlying best-effort network so as to match the requested QoS. We do so in a way that is consistent with the proper use of the network—namely, through the use of the Transmission Control Protocol (TCP) [3] for bandwidth allocation. Specifically, to maintain guaranteed bandwidth between any two points in the network, our approach calls for the establishment of an *elastic tunnel* between these points.[2] An elastic tunnel is simply a set of TCP connections between two points whose cardinality is dynamically adjusted in real-time so as to maintain a desirable target bandwidth. Typically, the end-points of this elastic tunnel would be edge routers within a single ISP, or in different ISPs; we call these edge routers *Internet Traffic Managers* (ITM). We refer to the set of TCP connections making up an ITM-to-ITM elastic tunnel as the *ITM-TCP connections* to distinguish them from user TCP connections originating and terminating at end-hosts. Figure 1 depicts the general model we consider throughout this paper.

Example Deployments: As we hinted above, elastic TCP-based tunnels could be established between ITMs within the same ISP, or between ITMs in different ISPs. Intra-ISP tunnels could be used as a mechanism to satisfy a certain Service Level Agreement (SLA) for a given customer on an existing best-effort (i.e. QoS-oblivious) network infrastructure. For example, an ISP with a standard best-effort IP infrastructure could offer its customers a service that guarantees a minimum bandwidth between specific locations (e.g., the endpoints of a Virtual Private Network (VPN) of an organization). Inter-ISP tunnels could be used as a mechanism to satisfy a desirable QoS (namely bandwidth) between two points without requiring infrastructural support from the ISPs through which such tunnels will go through (beyond simple accounting of the aggregate volume of traffic traversing the network).

Notice that for both intra-ISP and inter-ISP deployments, and since the underlying network infrastructure is assumed to be a common IP infrastructure, it is mandatory that the envisioned "elasticity" be implemented in a manner that will not trigger network mechanisms that protect against unresponsive flows (e.g., TCP unfriendly flows). In other words, to a core router, the constituent flows of an elastic tunnel must be indistinguishable from other TCP flows.

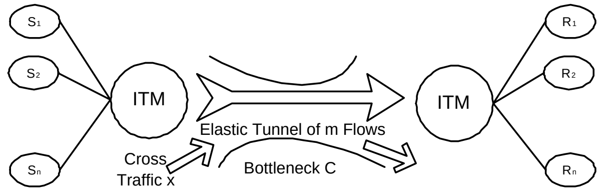


Figure 1: Elastic TCP-based Tunnel between ITMs

Without loss of generality, and for ease of presentation, in this paper we will focus on intra-ISP tunnels, with the understanding that all our results and observations are applicable to inter-ISP settings.

Overview of Model and Approach: Going back to our model in Figure 1, ITMs classify and manage traffic passing through them in a manner that is completely transparent from the original sending and receiving end-hosts. Consider n regular user connections between sending and receiving end-hosts, all passing through two ITMs. Again, one can think of these two ITMs as the gateways in a VPN, for example. Our main goal is to provide a soft-bandwidth-guaranteed tunnel for these user flows over an Internet path of bottleneck capacity C, which is also shared by another set of x flows, representing cross traffic. In this paper, we only consider user and cross-traffic connections to be TCP connections since TCP traffic is measured as constituting the majority of the bytes flowing over the Internet today [1]. These x cross-traffic connections present a challenge: as x keeps changing, the bandwidth allocation for the n user-TCP flows keeps changing in tandem. So an important question is whether it is possible to "counter" the change in x so as to ensure that the n user flows are able to maintain a desirable bandwidth.

Clearly without the intervention of ITMs, the answer to the above question is *no*. When different flows share a link, the effect of each individual flow (or an aggregate of flows) affects the rest since all are competing for a fixed amount of resources. However, if the ITMs dynamically maintain a number m of open TCP connections between them, they can provide a positive pressure that would equalize the pressure caused by the cross-traffic connections, if the latter occurs. Since m will be changing over time, we describe the ITM-to-ITM tunnel as *elastic*. Note that the source ITM can decide to reduce m (i.e. relieve pressure) if x goes down—the reason is that as long as the tunnel is achieving its target bandwidth, releasing extra bandwidth should improve

[2] Note that other performance metrics such as delay and loss can be controlled through these elastic soft-bandwidth-guaranteed tunnels.

the performance of cross-traffic connections, which is in the spirit of best-effort networking.

To illustrate our notion of elastic tunnels and the issues involved, consider an ITM-to-ITM tunnel going through a single bottleneck link. Under normal load, the behavior of the bottleneck can be approximated by Generalized Processor Sharing (GPS) [11], i.e. each TCP connection receives the same fair share of resources. Thus, each TCP connection ends up with $\frac{C}{m+x}$ bandwidth. This, in turn, gives the m ITM-TCP flows, or collectively the elastic ITM-to-ITM tunnel, a bandwidth of $\frac{Cm}{m+x}$. As the source ITM increases m by opening more TCP connections to the destination ITM, the tunnel can grab more bandwidth. If x increases, and the ITMs measure a tunnel's bandwidth below a target value (say B^*), then m is increased to push back cross-traffic connections. If x decreases, and the ITMs measure a tunnel's bandwidth above B^*, then m is decreased for the good of cross-traffic connections. It is important to note that the source ITM should refrain from unnecessarily increasing m, thus achieving a tunnel's bandwidth above B^*, since an unnecessary increase in the total number of competing TCP flows reduces the share of each connection and may cause TCP flows to timeout leading to inefficiency and unfairness [10]. The source ITM also has the responsibility of scheduling user packets coming on the n user connections over the tunnel, i.e. the ITM-TCP connections.

Paper Overview and Outline: In this paper, we develop a control-theoretic model to capture the flow-level dynamics of our model in Figure 1. We study the effect of different types on controllers as to optimize the transient behavior of established soft-bandwidth-guaranteed TCP-based tunnels. We confirm the premise of our approach by extensive ns-2 [5] simulations. We also outline our kernel-level ITM prototype implementation.

The rest of the paper is organized as follows. In Section 2, we present our design goals, and a detailed view of our proposed architecture and its basic components. In Section 3, we present a flow-level control-theoretic model focusing on the transient behavior of our elastic TCP-based tunnels. Section 4 defines our performance measures, describes our simulations through ns-2 [5], as well as discusses implementation issues. We revisit related work in Section 5. Section 6 concludes with a summary and future work.

2. Architecture for Elastic TCP-based Tunnels

In this section, we describe the main design goals and components of our architecture for soft-bandwidth-guaranteed tunnels.

2.1 Design Goals

The main goals of our architecture, in addition to providing soft bandwidth guarantees, are: (1) it should react to congestion signals from the network; (2) it should be friendly to other (cross-traffic) flows sharing network resources; (3) in case of severe congestion, it shouldn't over-react causing more congestion; and (4) it should minimize the creation and termination of the ITM-TCP connections that constitute the elastic tunnel.

Such goals are crucial for a healthy operation of the Internet. Often times, such goals would be conflicting. For example, due to the nature of cross-traffic flows, a degradation of the currently measured tunnel's bandwidth could occur. This requires an increase in the number of ITM-TCP connections, however, this reaction to such degradation should be smooth so as to balance the conflicting goals of maintaining the target bandwidth and being mindful of current congestion conditions.

Since our approach uses regular TCP flows between the ITMs, these flows react to congestion signals in the same way as regular TCP connections, thus serving our first two goals. Severe congestion situations could be detected, for example, by the tendency of TCP flows to timeout or to adapt to a small transmission window size. Obviously we would like to avoid such severe congestion conditions early before they happen. Thus, our third goal is achieved through preventing any creation of new ITM-TCP connection if existing ones are observed to have their congestion windows dropping below a certain threshold w_{min}. This in turn will prevent timeouts from occurring to the ITM-TCP flows that constitute the elastic tunnel, as well as avoid causing more congestion in the network. Finally, our fourth goal is achieved through careful adjustment of the transient behavior of our elastic tunnels. We discuss in more detail this transient performance when discussing the design of the controller component of our architecture.

2.2 Soft-Bandwidth-Guaranteed Tunnels

For the purpose of providing soft-bandwidth-guaranteed tunnels, the following components should be implemented at the origin ITM:

Monitor: The monitor component tracks the bandwidth grabbed by the elastic TCP-based tunnel established between the origin ITM and the destination ITM. This is calculated through measuring the rate of received bytes for all m ITM-TCP connections. The monitor measures the bandwidth over a measurement period (MP). Since MP should be large enough to capture the average behavior of the aggregate throughput, it should be in the order of few congestion epochs.[3]

Controller: Based on the error signal between the current bandwidth allocation grabbed by the m ITM-TCP flows and the desired bandwidth target B^*, the origin ITM invokes a controller which adjusts the number of open connections, by closing existing ITM-TCP connections or opening new connections or keeping them unchanged. Every time the controller is invoked, the monitor is queried for the measured tunnel's bandwidth. The controller is invoked every control period (CP), which we take to be equal to MP. We discuss the performance of different types of controller in Section 3.

Scheduler: The scheduler component is responsible for allocating the bandwidth acquired by the elastic TCP-based tunnel among the n user-TCP flows. Many scheduling policies can be used, e.g. WFQ [11]. Through the use of a WFQ scheduling algorithm, we can provide a weighted fair allocation of the achieved tunnel's bandwidth among different hosts and their applications, thus meeting their QoS requirements. This weighted allocation of tunnel's bandwidth and its implication of application performance is outside the scope of this paper. Our main focus here is how to achieve a desired tunnel's bandwidth, rather than how to re-allocate it among individual user flows. The scheduler is called on every user packet arrival.

Overall Architecture: Figure 2 shows the basic components for each entity in our architecture—the TCP/IP stack is shown for the sending and receiving end-hosts, and the origin and destination ITMs.

When a user's packet crosses the origin ITM, it first passes through the IP layer. If the packet belongs to a customer of the established elastic tunnel, the origin ITM removes the IP packet from the TCP/IP stack and places it in the scheduler buffer waiting to be scheduled over the m ITM-TCP flows.[4] The origin ITM elastic-tunnel application, on a non-empty scheduler buffer, would take the packet and send it out over one of the m ITM-TCP connections. To do so, the IP packet is encapsulated inside a TCP packet, which in turn gets encapsulated inside a new IP packet where the source IP address is that of the origin ITM and the destination IP address is that of the destination ITM.

Upon the receipt of an IP packet destined to the destination ITM, the packet passes normally through the TCP/IP stack, removing the IP header and then removing the TCP header. Thus, the user's original IP packet is delivered to the elastic-tunnel application running on the destination ITM. The elastic-tunnel application then passes the packet directly to the IP layer to be sent out. The exact Application Layer Interface (API) is described in Section 4.

It is worth noting that different techniques (e.g. zero copy [4]) exist to overcome inefficiencies of crossing stack boundaries. We report on the performance of ITM kernel-level implementations in another paper. In this paper, we focus on the evaluation of different controllers for the elasticity of the ITM-to-ITM tunnel and its effectiveness in providing a soft bandwidth guarantee.

3. Control-theoretic Analysis

In this section, we develop a control-theoretic model of different controllers employed at an origin ITM. Such controller determines the degree of elasticity of ITM-to-ITM TCP-based tunnels, thus it determines the transient and steady-state behavior of our soft-bandwidth-guaranteed service.

Naive Control: This naive controller measures the bandwidth b' grabbed by the current m' ITM-TCP connections. Then, it directly computes the quiescent number $\hat{m}$ of ITM-TCP connections that should be open as:

$$\hat{m} \quad = \quad \frac{B^*}{b'} m' \tag{1}$$

Clearly, this controller naively relies on the previously measured bandwidth b' and adapts without regard to delays in measurements and possible changes in network conditions, e.g. changes in the amount of cross traffic. We thus investigate general well-known controllers which judiciously zoom-in toward the target bandwidth value. To that end, we develop a flow-level model of the system dynamics. The change in the bandwidth grabbed $b(t)$ by the $m(t)$ ITM-TCP flows (constituting the elastic ITM-to-ITM tunnel) can be described as:

$$\dot{b}(t) \quad = \quad \alpha[(C - B^*)m(t) - B^* x(t)] \tag{2}$$

Thus, $b(t)$ increases with $m(t)$ and decreases as the number of cross-connections $x(t)$ increases. α is a constant that represents the degree of multiplexing of flows and we chose it to be the steady-state connection's fair share ratio of the bottleneck capacity. At steady-state, $\dot{b}(t)$ equals zero, which yields:

$$B^* \quad = \quad \frac{C\hat{m}}{(\hat{x} + \hat{m})} \tag{3}$$

where $\hat{m}$ and $\hat{x}$ represent the steady-state values for the number of ITM-TCP and cross-traffic flows, respectively.

[3] A congestion epoch is defined to be the time it takes for the transmission window of a TCP connection to linearly increase until a packet loss due to congestion occurs [8]. A multiple of congestion epochs should then reflect an average long-term behavior rather than capturing short-term fluctuations in sending rate.

[4] If the packet does not belong to a customer of an established elastic tunnel, the packet would be normally routed to its next IP hop based on its destination IP.

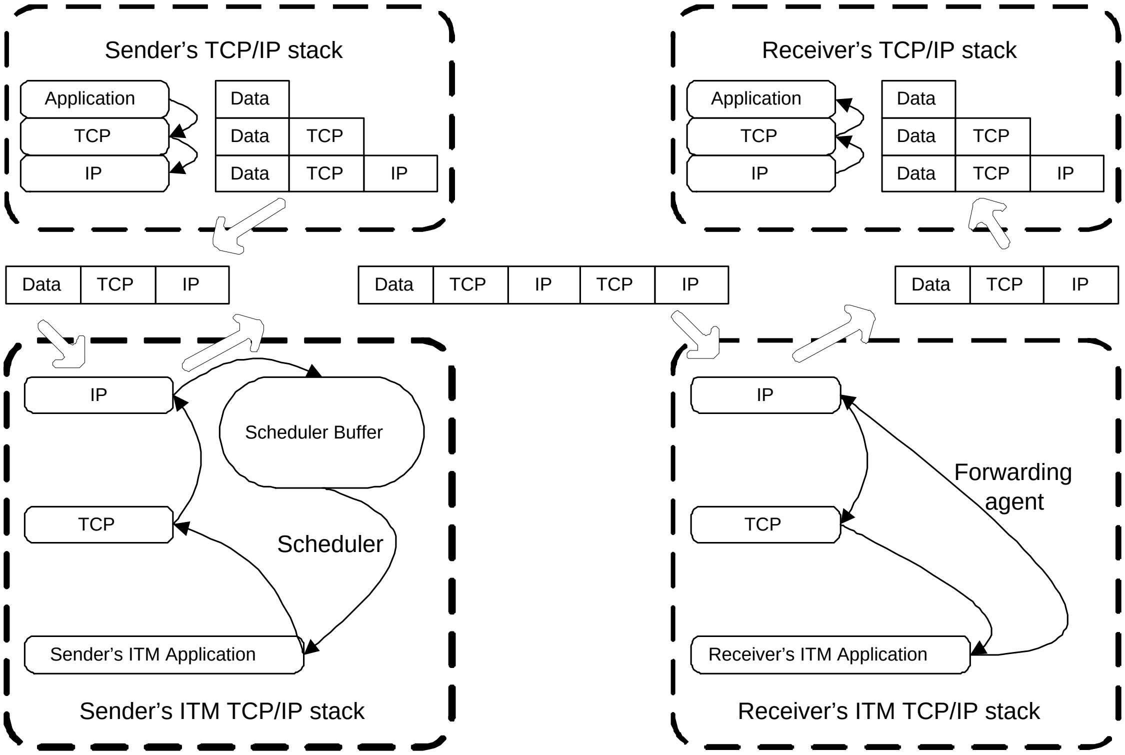


Figure 2: Overall Architecture for Elastic TCP-based Tunnels

Based of the current bandwidth allocation $b(t)$ and the target bandwidth B^*, an error signal $e(t)$ can be obtained as:

$$e(t) \quad = \quad B^* - b(t) \tag{4}$$

P and PI Control: A controller would adjust $m(t)$ based on the value of $e(t)$. For a simple Proportional controller (P-type), such adjustment can be described by:

$$m(t) \quad = \quad K_p e(t) \tag{5}$$

P-type controllers are known to result in a non-zero steady-state error. To exactly achieve the target B^* (i.e. with zero steady-state error), a Proportional-Integral (PI-type) controller can be used:

$$m(t) \quad = \quad K_p e(t) + K_i \int e(t) \tag{6}$$

Figure 3 shows the block diagram of our elastic-tunnel model. In the Laplace domain, denoting the controller transfer function by $C(s)$, the output $b(s)$ is given by:

$$b(s) = \frac{C(s)G_1(s)}{1 + C(s)G_1(s)} B^*(s) + \frac{G_2(s)}{1 + C(s)G_1(s)} x(s) \tag{7}$$

where $G_1(s)$ is given by:

$$G_1(s) \quad = \quad \frac{\beta}{s} \tag{8}$$

where $\beta = \alpha(C - B^*)$. $G_2(s)$ is given by:

$$G_2(s) \quad = \quad \frac{-\alpha B^*}{s} \tag{9}$$

where $\gamma = -\alpha B^*$. For the P-controller, from Equation (5), $C(s)$ is simply K_p. For the PI-controller, from Equation (6), $C(s)$ equals $K_p + \frac{K_i}{s}$. Thus, the transfer function $\frac{b(s)}{B^*}$ in the presence of a P-controller is given by:

$$\frac{b(s)}{B^*} \quad = \quad \frac{K_p \beta}{s + K_p \beta} \tag{10}$$

The system with P-controller is always stable since the root of the characteristic equation (i.e. the denominator of the transfer function) is negative, given by $-K_p\beta$. In the presence of a PI-controller, the transfer function $\frac{b(s)}{B^*}$ is given by:

$$\frac{b(s)}{B^*} \quad = \quad \frac{K_p \beta s + K_i \beta}{s^2 + K_p \beta s + K_i \beta} \tag{11}$$

One can choose the PI-controller parameters K_p and K_i to achieve a certain convergence behavior to the target bandwidth B^*. We next define transient performance measures to assess such convergence behavior.

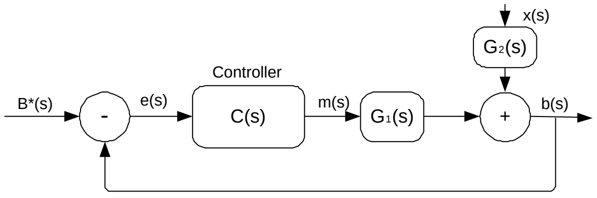


Figure 3: Block Diagram of Our Elastic-Tunnel Model

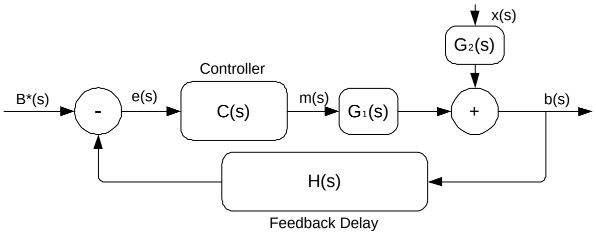


Figure 5: Elastic-Tunnel Model (with Feedback Delay)

3.1. Transient Performance Metrics

Transient behavior represents the system's response which decays with time. In the design of reliable systems, it is of extreme importance that transient response meets certain requirements such as reasonable settling time and overshoot. Often times, the transient response is obtained by subjecting the system to an impulse or a step input and observing the output(s). One has to guarantee that the response of the system to specific inputs does not render the system unstable or pushes it away from its intended target. For our specific elastic TCP-based tunneling system, we define our performance metrics as follows:

- **Settling Time:** The time taken for our system to respond to a step input in the cross traffic or target bandwidth until it stabilizes once again.

  The system is assumed to have stabilized (in steady state) if the error (the difference between target and measured bandwidth) is bounded for at least 2 seconds. Specifically, if $e(t)$ is the error at time t then the system is considered in steady state if

  $$\forall t \in [t_0, t_k] \text{ where } t_k - t_0 \geq 2\text{secs}, \bar{b} - \delta \leq e(t) \leq \bar{b} + \delta,$$

  where $\bar{b}$ is the average bandwidth measured during this period and δ is a constant. We chose $\delta = \frac{B^*}{20}$.

- **Maximum Overshoot:** The largest overshoot value experienced by the controller in terms of extra ITM-TCP connections opened or extra bandwidth allocated.

- **Stability in Number of ITM-TCP Flows:** The variability in number of ITM-TCP flows reflects the overhead of setting up and tearing down ITM-TCP connections within the elastic tunnel.

3.2. Transient Performance Results

Figure 4 shows the step response of the transfer function given in Equation (7). The left column shows the response to a step change in the target bandwidth, while the right column shows the response to a step change in the cross-traffic. Figure 4(a), for the P-controller, shows that while the response could be acceptable due to a step change in the reference bandwidth, it fails to remove the steady-state error (non-zero amplitude) obtained from the step change in the cross-traffic. Figures 4(a) and (b) show the response due to the PI-controller. One can see that through a careful choice of K_p and K_i, the transient response can be adjusted. Notice that with a PI-controller, our elastic-tunneling system can reach the target bandwidth with zero steady-state error in response to a step change in cross-traffic.

3.3. Feedback Delay

So far in our analysis, we have ignored the feedback delay which is inherent in the design of any control system that tries to adjust its signal through a delayed feedback loop.

Figure 5 augments the block diagram of Figure 3 with feedback delay denoted by $H(s)$. This feedback delay arises either due to delayed mesurements of bandwidth and/or delayed response of the system as a result of applying new control. For example, when a new ITM-TCP connection is opened, it doesn't get its steady-state throughput instantaneously, rather after some delay (say τ). Thus, $H(s)$ is given by:

$$H(s) = e^{-\tau s} \tag{12}$$

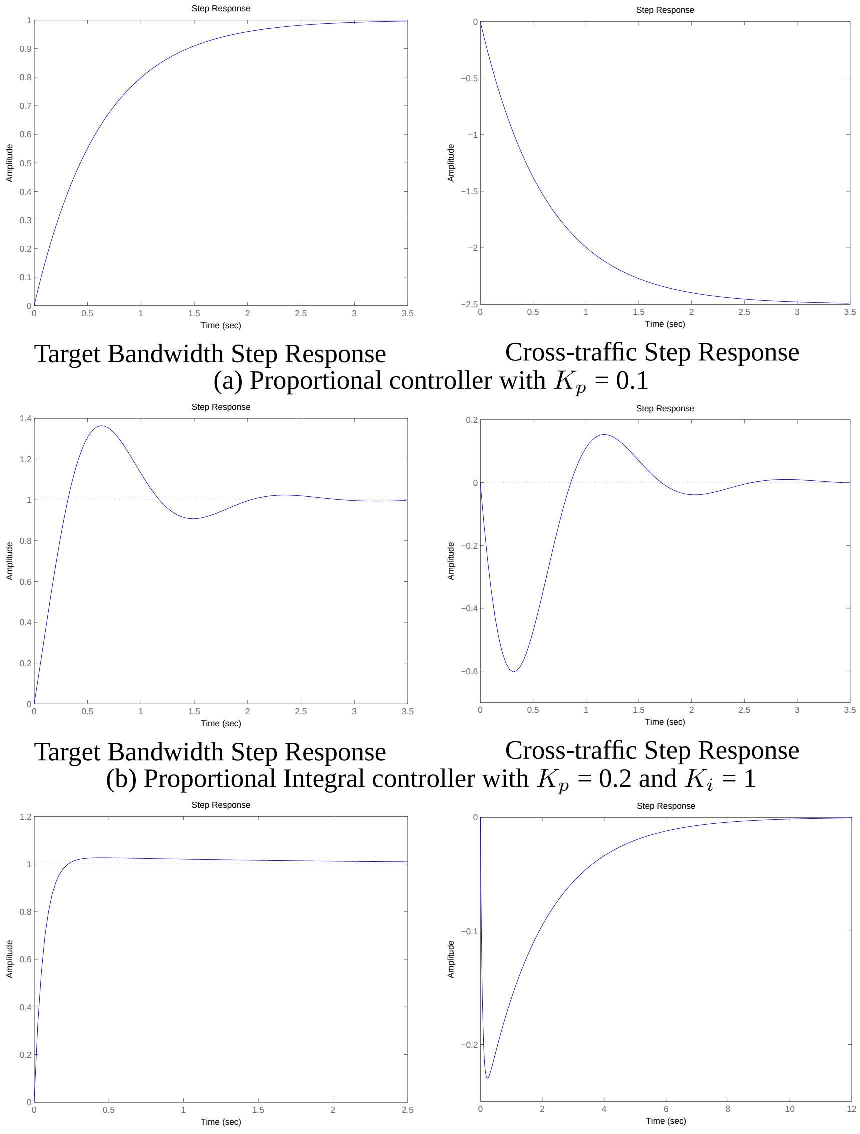
where τ represents the feedback time delay. For small τ, the above equation can be approximated by:

$$H(s) = 1 - \tau s \tag{13}$$

If we are using a PI-controller, the characteristic equation in the presence of feedback delay becomes:

$$s^2(1 - \beta\tau K_p) + s(K_p\beta - \beta\tau K_i) + \beta K_i \tag{14}$$

Figure 6 shows the response of the PI-controller to a step change in the target bandwidth. As the feedback delay τ increases, the system may not converge to the target bandwidth.



Target Bandwidth Step Response Cross-traffic Step Response

(a) Proportional controller with K_p = 0.1

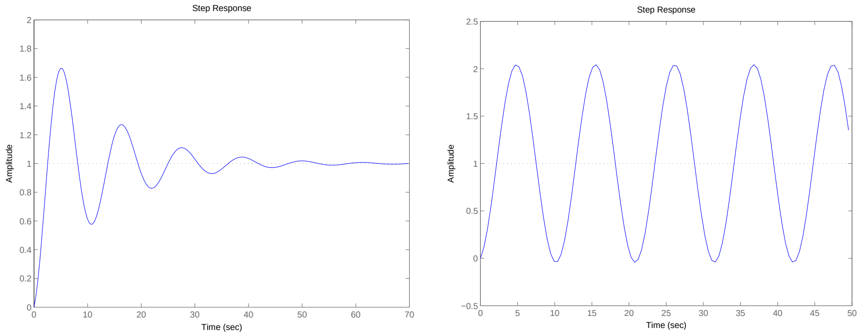
Target Bandwidth Step Response Cross-traffic Step Response

(b) Proportional Integral controller with K_p = 0.2 and K_i = 1

Target Bandwidth Step Response Cross-traffic Step Response

(c) Proportional Integral controller with K_p = 1 and K_i = 0.5

Figure 4: Transient Analysis of our Elastic-Tunnel Model



Target Bandwidth Step Response Response with Feedback Delay

(d) Proportional Integral controller with K_p = 0.01 and K_i = 0.02

Figure 6: Transient Analysis in the presence of Feedback Delay

4. Simulation Results

In this section, we present results from extensive ns-2 [5] simulation experiments. These results confirm our analysis and demonstrate the effectiveness of our proposed architecture in establishing elastic soft-bandwidth-guaranteed tunnels. We then expand Section 2 by outlining more details on our kernel-level ITM prototype implementation.

4.1. Simulation Experiments

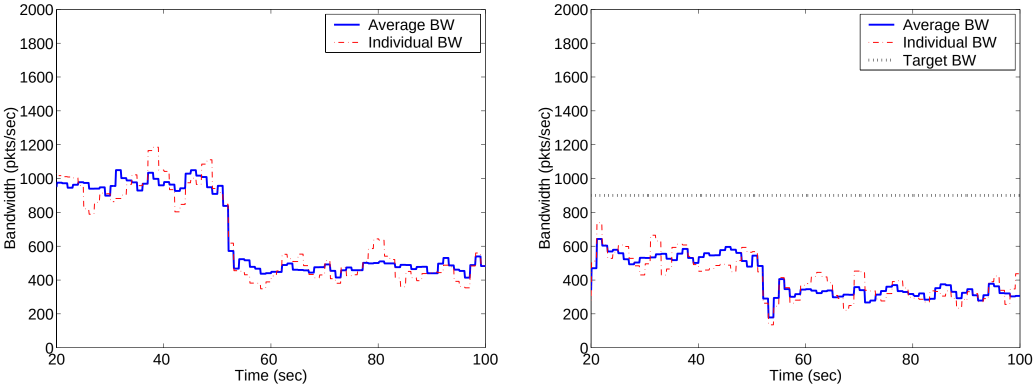
Topology Setup: Figure 1 depicts the topology under consideration. The bottleneck link has 16Mb/s (2000 $\frac{pkts}{sec}$) capacity and a 2-ms one-way propagation delay. We vary the propagation delay on the access links so different flows have different round-trip times. The bottleneck link is shared between ITM-TCP connections and cross-traffic connections and employs RED queue management [6].[5] All connections are considered to have unlimited data to send and they all use TCP Reno. The buffer size is chosen to be 250 packets. All packets are 1000 bytes in size. RED's minimum and maximum buffer thresholds are set to 50 and 120 packets, respectively. The RED's weight parameter was set to 0.0001 and P_{max} was set to 0.1. We focus on the transient behavior of different controllers. We ignore the first 20 seconds of the simulation time as a warm-up period. The Measurement Period (MP) as well as the Control Period (CP) are chosen to be 2 seconds.

Experiment 1: Our first experiment illustrates the challenges we are faced with when we don't exercise any control over the number of ITM-TCP connections between the ITMs. This scenario resembles the case when the ITMs are not present and the user TCP connections are left to compete for themselves against cross-traffic. We start our experiment with 10 ITM-TCP connections sharing the bottleneck along with 10 cross-traffic TCP connections. At time 50, we introduce a step increase of 20 in the number cross-traffic connections—that is an increase from 10 connections to 30 connections. Figure 7(a) shows the aggregate bandwidth acquired by the ITM-TCP connections. We plot both, the bandwidth allocated for a single simulation run as well as the average bandwidth allocated averaged over 5 independent runs. One can see that such a step increase in the number of cross-traffic connections degrades the total bandwidth allocated to the ITM-TCP connections. This is exactly the response of an open-loop system where the number of ITM-TCP connections is not adapted in response to changes in the amount of cross-traffic.

Experiment 2: In this experiment, we investigate the use of the Proportional Controller. With the same setup as experiment 1, the soft bandwidth target is set to 900 $\frac{pkts}{sec}$ and K_p is chosen to be 0.01. Figure 7(b) shows the aggregate bandwidth acquired by the ITM-TCP connections. One can see that the P-controller fails to adjust the bandwidth of the elastic tunnel to the target value. Moreover, it never eliminates the steady-state error, either before or after applying the step increase in the number of cross-traffic connections. This confirms our analysis shown in Figure 4(a).

Experiment 3: We repeat the same above experiment except that we use the naive controller to adapt the number of ITM-TCP connections. Figure 8(a) shows the aggregate bandwidth acquired by the ITM-TCP connections when subjected to an increase in the (desired) target bandwidth from 900 $\frac{pkts}{sec}$ to 1200 $\frac{pkts}{sec}$ at time 50. Figure 8(b) shows the case of a step increase in the number of cross-traffic connections from 10 to 30 at time 50. Since the naive controller tries to adjust the number of ITM-TCP connec-

[5] We note that our elastic-tunnel service does not require any specific queue management policy. Specifically, core routers may use simple FCFS (First-Come-First-Serve) queues. In practice, randomization comes from unsynchronized arrivals/departures of flows/packets and FCFS queues would serve TCP flows in a processor-sharing fashion, giving each flow its fair share of the resources.



(a) Cross-traffic Step Response

(b) Cross-traffic Step Response

Figure 7: (a) No Control is applied; (b) Proportional Controller is used.

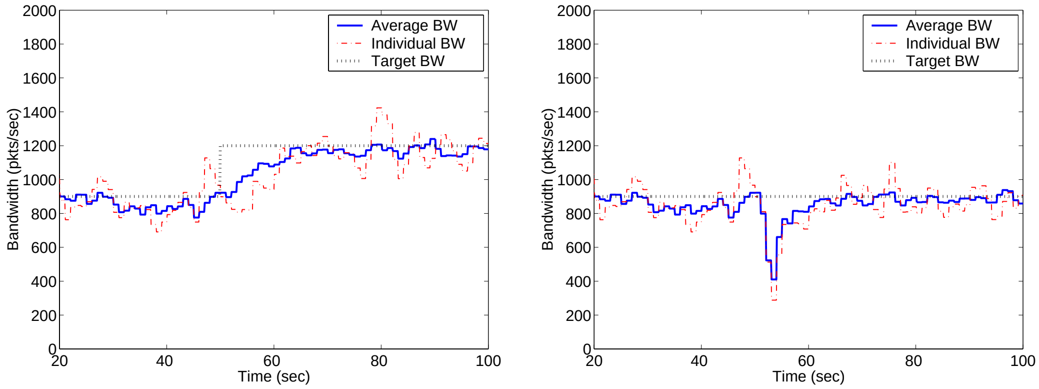
tions in a single step of control, its behavior is rather instantaneous. One can see how it can adapt quickly to the change in the target bandwidth and/or the number of cross-traffic connections. While such behavior may seem appropriate for general use of the naive controller, this controller has its own drawbacks. In particular, it is problematic in cases where there is a significant and continuous change in network conditions. This is the subject of our next experiment.

Experiment 4: The purpose of this experiment is to demonstrate the drawbacks of the naive controller through a more dynamic behavior of cross-traffic connections. Specifically, cross-traffic connections start and stop sending data every 10 seconds starting at time 50. This has the effect of a square signal in the data sent by the cross-traffic. Figure 9(a) shows the behavior of the elastic tunnel under these square signals in the cross-traffic. Figure 9(b) shows the number of open ITM-TCP connections at any instant of time. As illustrated, the naive controller doesn't try to stabilize the number of open ITM-TCP connections which is one of our main design goals. Rather the naive controller tends to open and close a large number ITM-TCP connections at every control period. Figures 9(c) and (d) show the behavior of the PI-controller. One can see that it opens less ITM-TCP flows than the naive controller. The maximum number of open ITM-TCP connections at any time was 20 as opposed to 40 connections for the naive controller. Also, the bandwidth acquired by the elastic ITM-to-ITM tunnel tends to oscillate less than in the naive controller case.

Experiment 5: As our previous control-theoretic analysis (cf. Section 3) has indicated, through adjusting the values of K_p and K_i, the overall behavior of the PI-controller can be changed. In this experiment, we point out the two cases of underdamped and overdamped responses shown by analysis in Figures 4(b) and (c), respectively. As in previous experiments, we subject the elastic ITM-to-ITM tunnel (i.e. the constituent ITM-TCP connections) to a step increase in the target bandwidth from 900 $\frac{pkts}{sec}$ to 1200 $\frac{pkts}{sec}$, and a step increase in the number of cross-traffic TCP connections from 10 to 30, both applied at time 50. Figure 10 (top row) depicts the overdamped case (K_i was chosen to be 0.01) while Figure 10 (bottom row) depicts the underdamped case (K_i was chosen to be 0.08). Figure 10(c) and (f) zoom-in at the interval of time between 50 and 70. We observe the exact same trends as shown by analysis in Figures 4(c) and (b), respectively (overdamped case and underdamped case). For the underdamped case (bottom row), having a larger value of K_i when multiplied by the error signal tends to make the PI-controller more aggressive in terms of the number of ITM-TCP connections to open or close. That is why it tends to be in an unstable (limit-cycle) regime before time 50. Interestingly enough, this trend matches the analysis we provided in Figure 6 in the presence of feedback delay. When the system is underdamped, it tends to overshoot, however if feedback delay is present, it tends to drive the system into a limit cycle regime. After time 50, the oscillations are reduced significantly. This is due to the decrease of the gain of the whole system since now each connection fair share is less due to the increase in the number of competing TCP connections. This experiment shows that an overdamped system (first row) is a good choice to be used in practice—it behaves gracefully and doesn't tend to overshoot.

Another Setup: For the following two experiments we change the bottleneck link capacity to 50 Mb/s (6250$\frac{pkts}{sec}$). The round-trip propagation delay was chosen to be 100 msec. The Measurement Period (MP) as well as the Control Period (CP) are chosen to be 1 second. The results show that the effectiveness of our elastic-tunnel approach is insensitive to different bandwidth/delay values.

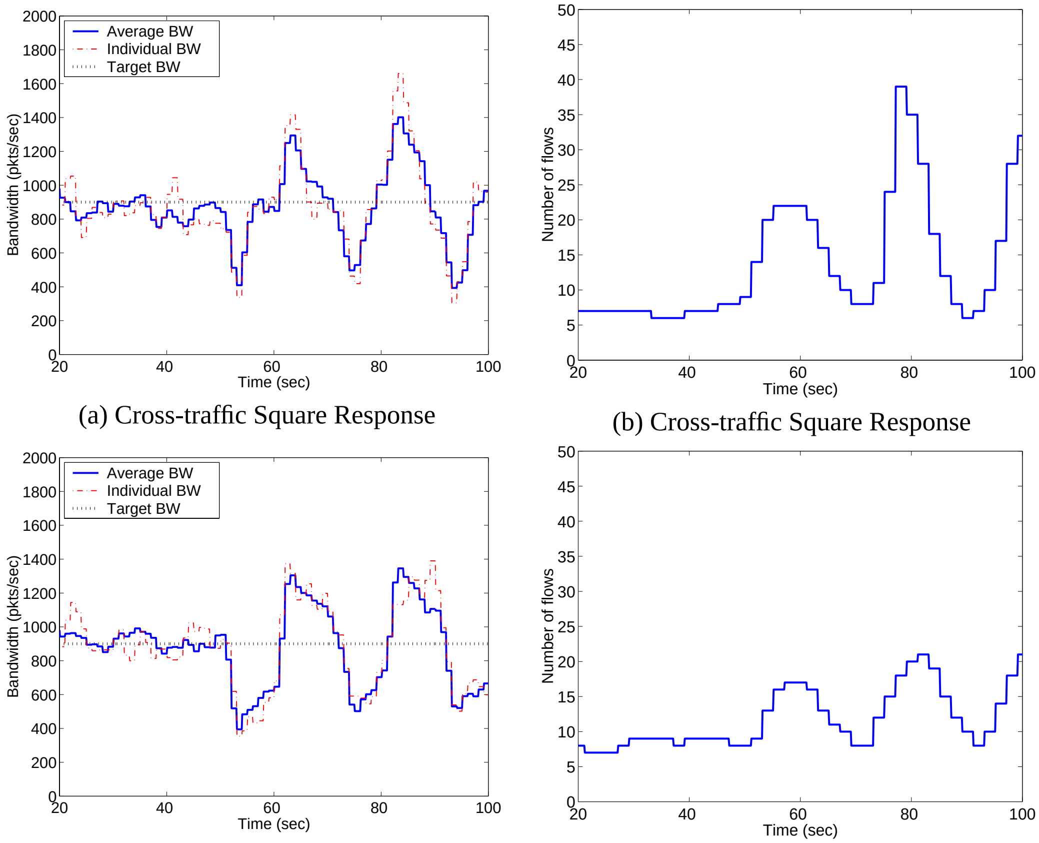
Experiment 6: In this experiment, we start with a target bandwidth of 1875 $\frac{pkts}{sec}$. At time 30, the target bandwidth



(a) Target Bandwidth Step Response

(b) Cross-traffic Step Response

Figure 8: Naive Controller



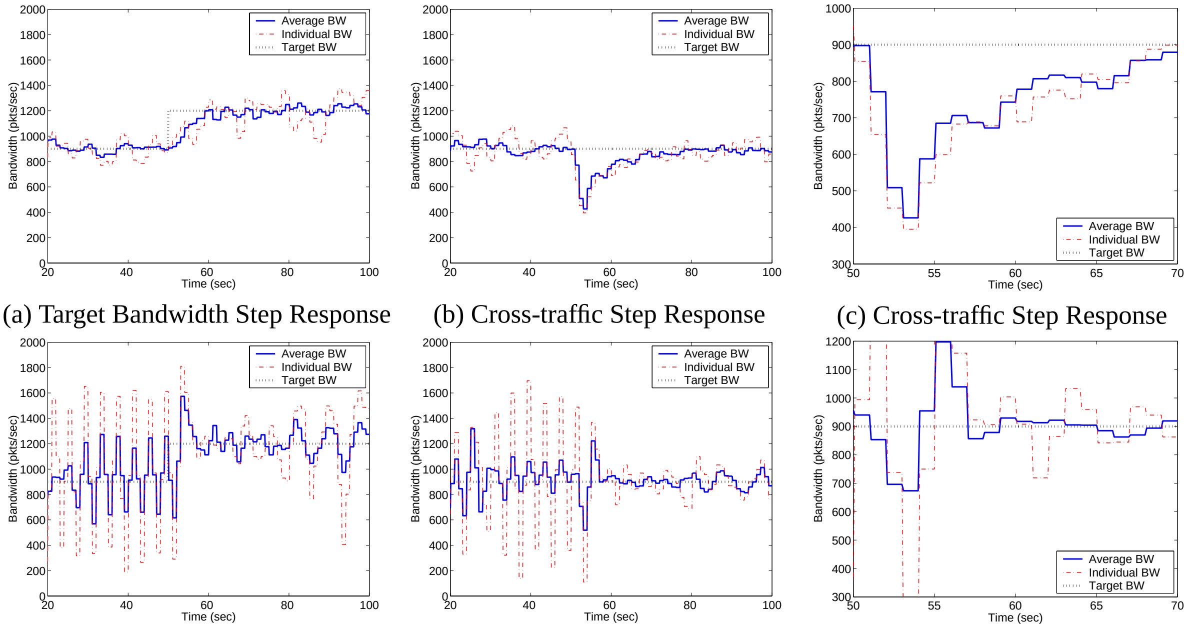
(a) Cross-traffic Square Response

(b) Cross-traffic Square Response

(c) Cross-traffic Square Response

(d) Cross-traffic Square Response

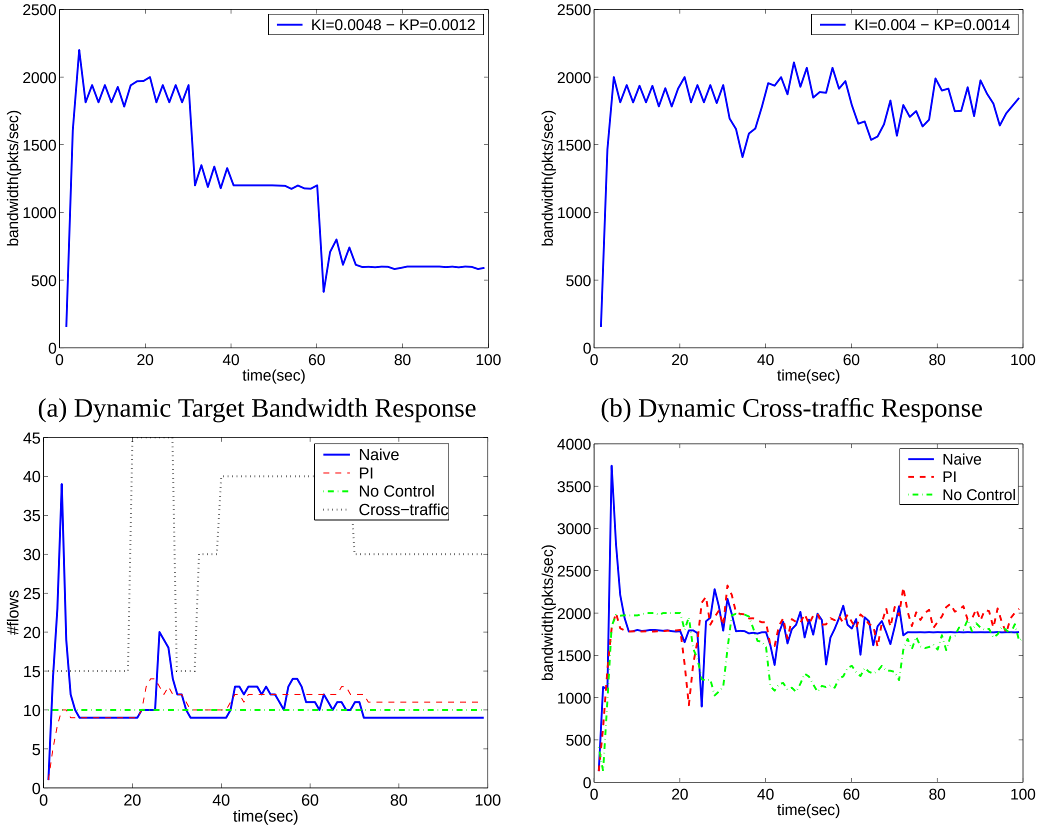
Figure 9: (Top row) Effect of dynamic cross-traffic on the Naive Controller; (Bottom row) Effect of dynamic cross-traffic on the PI Controller



(a) Target Bandwidth Step Response (b) Cross-traffic Step Response (c) Cross-traffic Step Response

(d)Target Bandwidth Step Response (e) Cross-traffic Step Response (f) Cross-traffic Step Response

Figure 10: Proportional Controller (first row) K_i is set to 0.01 and (second row) K_i is set to 0.08



(a) Dynamic Target Bandwidth Response (b) Dynamic Cross-traffic Response

(c) Number of open ITM-TCP flows (d) Aggregate acquired Bandwidth

Figure 11: Results for different controllers in a more dynamic environment

decreases to 1250 $\frac{pks}{sec}$. At time 60, it is further decreased to 625 $\frac{pks}{sec}$. The cross-traffic is static. Figure 11(a) shows the bandwidth acquired by our elastic tunnel with a PI-controller, which is seen to adapt very well.

We repeat the experiment, this time changing over time the number of cross-traffic flows. In particular, we start with 10 cross-traffic flows; at time 30, the number of cross-traffic flows is increased to 30; and finally at time 60, it is increased to 50. The target bandwidth is static. Figure 11(b) shows how the PI-controller stabilizes the system as expected from the analysis of Section 3.

Experiment 7: Here, we move to a more dynamic environment. In this experiment, we compare the naive controller, the PI and the case where no control is applied. In this scenario we change the cross-traffic over time as shown in Figure 11(c). When no control is applied, as one would expect, the aggregate bandwidth obtained by user-TCP flows is very sensitive to changes in the cross-traffic (Figure 11(d)). The naive controller, despite the high overshoots, finally stabilizes the system. Figure 11(c) shows the oscillating behavior of the naive controller. It is undesirable to erratically open and close ITM-TCP connections and therefore this controller is not an optimal choice for highly dynamic environments. On the contrary, the PI-controller stabilizes the achieved bandwidth around the desired target in a less aggressive (smooth) manner.

4.2. Kernel Level Implementation

We implemented an event-driven API to our ITM architecture. Applications (both kernel and user level) wishing to use the ITM need to register with a core kernel module, providing pointers to five functions: (1) a *classification* function that specifies how to identify a class of packets or flows; (2) a *logging* function that specifies how to log data and update class structures; (3) a *processing* function that determines the action associated with each event; (4) a *class-update* function which defines how a class should be updated based on the logged data; and (5) a *controller-update* function which allows for updating the controller parameters based on system state. These functions are all called in this order, when the event an application registered for occurs. Events can either be synchronous (e.g. packet arrival at a specific layer) or asynchronous (e.g. periodic). In the case of our elastic-tunnel application studied in this paper, for example, asynchronous events occur due to periodic updates of the acquired tunnel's bandwidth measured by the monitor component and periodic updates of the number of tunnel's constituent ITM-TCP connections as determined by the controller component.

The base of the system is an ITM kernel module, which communicates with the TCP/IP stack to retrieve packets. The module keeps a list registered applications, and would forward packets to them according to their specifications. A control utility enables adding kernel modules and user-level components, each can be turned on or off according to need. This design results in a system that is extensible and easy to deploy. Preliminary results from our Linux-based prototype confirm the basic premise of our elastic-tunnel application. Figure 12 shows how the allocated bandwidth drops at time 20 with increasing cross traffic. This drop triggers the (naive) controller to create new ITM-TCP management connections to reach the desired target bandwidth. We report on the full performance of our elastic-tunnel application within this generalized ITM-API in a future paper.

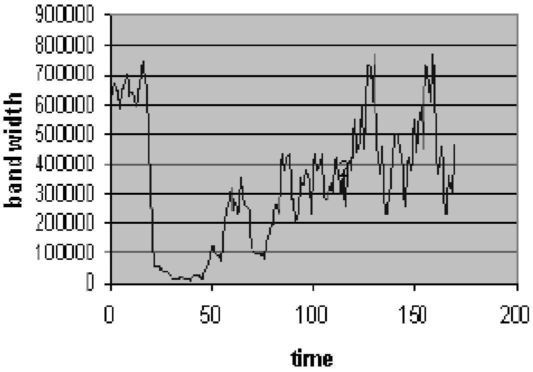


Figure 12: Controller adapts to Target Bandwidth on Linux Prototype

5. Related Work

QoS Frameworks: Providing Quality-of-Service (QoS) to data flows has been one of the important and challenging problems of interest to the Internet research community. Proposed QoS frameworks can be taxonomized along a number of dimensions based on (1) the type of guarantees they extend, e.g., hard versus soft, (2) the scalability of the solution, (3) the infrastructural network support needed, e.g., core router functionalities, and (4) the compatibility with best-effort services. IntServ [2] frameworks are capable of providing hard guarantees to data flows. This comes at the cost of maintaining per-flow state at every router along the path. The complexity of such architectures raises scalability concerns and some see such complexity contradicting the intended open architecture of the Internet. With the correct admission control and policing, IntServ frameworks could co-exist with other solutions as well as other best-effort services. On the other hand, DiffServ [15] frameworks are capable of providing soft guarantees. They still require some support from the network and they tend to provide a more scalable design that is more consistent with the scalable design philosophy of IP, pushing functionality toward the end-hosts. However it is unclear how well such frameworks would interact with other solutions as well as other best-effort services. Like DiffServ, our framework achieves soft guarantees. Contrary to DiffServ, our framework doesn't require any network support. Moreover, our proposed framework can co-exist with other solutions, since it is based on a TCP friendly application. It is completely transparent to end-systems.

End-to-End Adaptation: Other works have focused on developing end-system protocols that try to adapt the resources provided by the network to the needs of the application. For example, some studies (e.g., [8]) proposed different control rules for TCP behavior. By applying the right control rule, other properties can be achieved such as smoothness, aggressiveness and convergence, while maintaining friendliness to co-existing TCP traffic. This is particularly useful for streaming, real-time and gaming applications. Other studies (e.g., [9, 13, 12]) proposed that modification in transmission control rules be done on aggregates rather than individual flows, with the notion of flows sharing congestion information. For example, in [9], congestion information from a separate management connection (or using an architecture such as the Congestion Manager [13]) is used to regulate the aggregate traffic. Other techniques, such as Aggregate TCP (ATCP) [12] provides a congestion window lookup for an appropriate window size for new connections to start with. However, none of these techniques considered providing flows with a guaranteed service, but rather making flows adapt to available resources more adequately.

6. Summary

We presented a framework for providing soft bandwidth-guarantees over a best-effort network. Such a guarantee is provided through the use of an elastic TCP-based tunnel running between ITMs. The target bandwidth could be dynamically adjusted to meet the needs of applications. The elasticity of the established tunnel is achieved by adjusting the number of open TCP connections between ITMs to a quiescent number, large enough to push back against cross-traffic. This is performed in a completely transparent way from the sending and receiving end-hosts. Moreover, our framework allows for the QoS support of individual applications by preferentially allocating the bandwidth provided by the established elastic tunnel.

We presented simulation results showing the effectiveness of our approach in allocating the target bandwidth. Moreover, our approach remains responsive to congestion and degrades gracefully in severe congestion cases. Implementation details are also discussed and preliminary results on our Linux prototype are consistent with our control-theoretic analysis and simulations.

Acknowledgment: We would like to thank Sean Chen and Leonid Veytser for their contributions to the kernel-level implementation, and Rich West for his feedback.

References

[1] Measurement studies of end-to-end congestion control in the internet. http://www.icir.org/floyd/ccmeasure.html.

[2] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. *RFC 1633*, June 1994.

[3] V. Cerf and L. Kahn. A Protocol for packet Network Interconnections. *IEEE Transactions on Communications*, (5), 1974.

[4] J. Chu. Zero-copy TCP in solaris. In *USENIX Annual Technical Conference*, pages 253–264, 1996.

[5] E. Amir et al. UCB/LBNL/VINT Network Simulator - ns (version 2). Available at http://www.isi.edu/nsnam/ns/.

[6] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[7] L. Guo and I. Matta. The War between Mice and Elephants. In *Proceedings of ICNP'2001: The 9th IEEE International Conference on Network Protocols*, Riverside, CA, November 2001.

[8] S. Jin, L. Guo, I. Matta, and A. Bestavros. A spectrum of TCP-friendly window-based congestion control algorithms. *IEEE/ACM Transactions on Networking (TON)*, 11(3), June 2003.

[9] H. Kung and S. Wang. TCP trunking: Design, implementation, and performance. In *Proceedings of IEEE ICNP*, November 1999.

[10] R. Morris. TCP behavior with many flows. In *Proceedings of IEEE ICNP*, Atlanta, GA, October 1997.

[11] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking (TON)*, 1(3), June 1993.

[12] P. Pradhan, T. Chiueh, and A. Neogi. Aggregate TCP congestion control using multiple network probing. In *Proceeding of the 20th International Conference on Distributed Computing System*, April 2000.

[13] H. Balakrishnan H. S. Rahul and S. Seshan. An integrated congestion management architecture for internet hosts. In *Proceedings of ACM SIGCOMM*, September 1999.

[14] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[15] S. Blake D. Black M. Carlson E. Davies Z.Wang and W. Weiss. An architecture for differentiated services. *IETF RFC 2475*, December 1998.