Traffic Shaping via Congestion Signals Delegation

Mina Guirguis
Computer Science Department
Texas State University - San Marcos
San Marcos, TX 78666, USA
Email: msg@txstate.edu

Jason Valdez
Computer Science Department
Texas State University - San Marcos
San Marcos, TX 78666, USA
Email: jv1150@txstate.edu

***Abstract*—This paper presents a new architecture that enables a set of clients to enforce traffic shaping policies among them through the delegation of congestion signals. When congestion-aware Internet flows share a bottleneck link, they compete for bandwidth and must respond to congestion signals promptly by decreasing their throughput. For clients running real-time applications (e.g., gaming, streaming), this may impose strict limitation on their achievable throughput over short time-scales. To that end, this paper presents an architecture, whereby a set of TCP connections (we refer to them as the Stunts) sacrifice/trade their performance on behalf of another TCP connection (we refer to it as the Free) by picking up a delegated subset of the congestion signals and reacting to them in lieu of the Free connection. This gives the Free connection just enough freedom to meet specific throughput requirements as requested by the application running on top, without affecting the level of congestion in the network. We present numerical model and analysis, which we validate by extensive simulation as well as through a pluggable module implementation for the Linux kernel.**

***Keywords*-Service-oriented architecture; TCP; Congestion Control; Traffic Shaping; Control Theory;**

I. INTRODUCTION

Motivation: Certain classes of applications (e.g., gaming, audio and video streaming) need to acquire/maintain certain guarantees in order to perform adequately. Due to the "best-effort" nature of the Internet, it is very difficult to ensure that these guarantees are met or even to predict what possible guarantees could be provided. Hence, these applications are often left with unspecified guarantees on their quality of service. Research efforts have addressed this problem and proposed two major architectures, Integrated Services (IntServ) and Differentiated Services (DiffServ). IntServ architectures require *every* router to maintain per-flow state. Applications make reservations based on their needs. The main problem with IntServ is that it does not scale to a size that is as large as the Internet. Thus, it is limited to small-scale deployments. DiffServ, on the other hand, push traffic management towards the edges of the network, while keeping its core simple. Edge routers maintain and classify flows based on classes. Core routers still need to maintain brief information on how to treat each class. In both architectures, some modifications had to be made to some routers.

The "Free and Stunts" Architecture: In this paper, we propose a new end-host service architecture that provides an application with *soft throughput guarantees* over a best-effort network, without *any* modification to routers. Moreover, this is achieved in a completely friendly manner to the network through strictly adhering to the Transmission Control Protocol (TCP) rules. In particular, we envision a set of TCP connections (we refer to them as the Stunts connections) that are willing to sacrifice their own performance on behalf of another TCP connection (we refer to it as the Free connection). This would enable the Free connection to match its throughput to the target throughput from the application. In [1], we have demonstrated the feasibility of this idea through simulations only. In this paper, we extend this work by introducing a dynamic model (along with numerical solutions) as well as real implementation of this architecture in the Linux kernel.

TCP employs congestion control mainly via the Additive Increase Multiplicative Decrease (AIMD) mechanism that seeks to constantly probe for available capacity while remaining fair to other TCP flows [2], [3]. Congestion signals (as in dropped/ marked packets) signal TCP senders to slow down, by halving their congestion windows. The quantity and the timing of these congestion signals may prevent the Free connection from achieving any throughput guarantees. The main idea behind our architecture is to allow the Free connection to delegate a subset of those congestion signals to the Stunt connections. Thus, the Stunt connections would be the ones that decrease their sending rates instead of the Free connection, which would be liberated (to a larger extent) to match the guarantees requested from the application above. It is important to note that this architecture *does not* increase network congestion at the bottleneck link because it ensures that the total decrease in throughput from all the Stunt connections is at least as large as what the Free connection would have decreased, if it were to observe those delegated losses. For example, it is possible for a single packet loss delegated from the Free connection to cause more than one Stunt connection to back-off, in order to have the same equivalent effect on the bottleneck link.

The choice of Stunt connections is important due the

dynamic nature of Internet traffic. It has been evident through many measurement studies that TCP connections can be characterized into long-lived and short-lived flows, also known as elephants and mice, respectively [4], [5], [6], [7]. Elephant flows, while there are few of them, they account for around 80% of the traffic. These are long-lived stable TCP flows. Mice, on the other hand, while there are plenty of them, they account for only around 20% of the traffic. These are short-lived flows (simple HTTP requests and they typically finish before leaving the slow-start phase of TCP [8], [9]). In our proposed architecture, we limit the choice of Stunt connections to elephants as they are more stable and control the majority of Internet traffic. Moreover, delegating losses to mice would hurt their performance significantly, if they were chosen to act as Stunts.

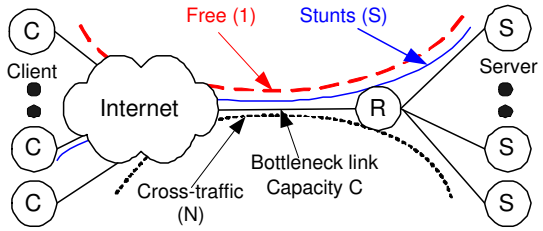

Figure 1. The Free and the Stunts setup. The proposed architecture is implemented at the End-host at the server's side.

Deployment Scenarios: This architecture is proposed to be used by Internet Service Providers (ISPs) to enforce differentiated service among its clients, by having some behave as the Free connections, while others play the Stunt roles. For example, by dropping appropriate packets (or marking them if the use of the ECN bit is enabled), an ISP (e.g., the first-hop router) can delegate losses between the Free and the Stunts. Such delegations could be based on a marketplace in which clients' agents agree upon what is fair and efficient for each one [10].

This architecture is also envisioned to be used by Internet servers that serve different forms of media content to different clients. With this architecture, a server can give a particular flow (say a media stream) the freedom to achieve requested guarantees, while making other flows (say long bulky file transfers, i.e., "elephants") behave as Stunts. In this scenario, the architecture is implemented at the end-host and thus the server can accurately identify the elephant flows based on the requested content (e.g., large files) from the clients. Figure 1 shows an example of such deployment where the server implements the proposed architecture to manage its first-hop to the Internet (which is the one that is typically prone to congestion).

Paper Organization: Section II puts this work in contrast to other related work. Section III describes our proposed architecture in more detail with all its components and logistics. Section IV captures the dynamics involved through numerical results based on a non-linear fluid model. We evaluate the performance of our proposed architecture with extensive simulation experiments in Section V. In Section VI we present results from our Linux implementation. We conclude the paper in Section VII.

II. Related Work

As hinted in the introduction, many Quality of Service (QoS) mechanisms have been proposed that belong to the general IntServ [11] or DiffServ [12] architectures. Due to their reliance on modifying some in-network components (whether core routers or edge routers), their acceptance for deployment is difficult. Moreover, some of them do require the participation of all entities, which imposes a significant scaling and implementation issues.

The work in [13] focused on managing the end-to-end behavior of TCP connections through sharing congestion information among them. A congestion manager module is used to regulate the transmission rates of the TCP connections in order to achieve an overall better performance. This method, however, does not aim to provide specific guarantees to the TCP connections. In [14] a coordination protocol (CP) is proposed which seeks to optimize cluster-to-cluster communication of computing devices across a bottleneck aggregation point. Their proposal entails, among other aspects, giving the flows across the aggregation point the ability to sense the network state and adapt at the end-points. The authors in [15] propose a Rate Management Protocol (RMP) that controls the rates of the flows passing through an aggregation point based on their QoS requirements. Once the fair share of the flows are decided by the RMP, a new TCP sliding window is used to realize that fair share. This method, however, requires the modification of the current architecture (aggregation and end points) to be realized. In [16], the authors divide the problem of managing QoS into two components; the first one utilizes a probing scheme at the IP layer and the other policies the rates at the edge routers. This method also requires the modification of edge routers.

In [17] an elastic tunnel is created using a number of regular TCP connections to provide soft bandwidth guarantees. The number of connections that form the elastic tunnel is adjusted dynamically in tandem with cross-traffic, so their aggregate throughput ensures the QoS guarantees. This work only considered constant QoS guarantees. Moreover, it required modifications to edge routes to manage the elastic tunnels.

The authors in [18] present an adaptive Forward Error Correction (FEC) scheme that aims to ensure a specific end-to-end rate for video streaming applications using TCP connections. The idea is to adjust the degree of redundancy in packets based on the difference between the achievable

throughput and the required rate, without wasting network bandwidth.

In addition to the above related work, we refer the readers to [19], [20] that provide surveys on bandwidth adaptation and control mechanisms for managing Internet traffic.

III. The Architecture

In this section, we describe the main components and the operation of our proposed service architecture.

A. The Components

We envision a setup composed of a single Free TCP connection and s Stunt TCP connections. Those $s+1$ TCP connections traverse a bottleneck link along with n other TCP connections representing normal cross-traffic. Thus a total of $(1+s+n)$ TCP connections traverse that bottleneck link. Figure 1 depicts this setup. The application running on top of the Free connection requests its throughput requirements through a trace file. This trace file is first checked by a preprocessor for feasibility. If any of the checks fail, another feasible trace is created that is closest to the original trace file; otherwise, the trace file is passed directly to the controller. The controller compares the achievable throughput to the requested throughput over every time instant. Based on the difference, the controller adjusts the ratio of congestion signals to be delegated from the Free connection to the Stunts in order to match the achievable throughput to the requested throughput. A monitor measures the throughput achieved by the Free connection and reports this value back to the controller. Figure 2 represents the different components in the architecture.

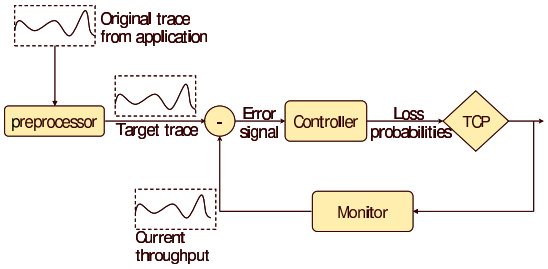

Figure 2. The Components of the proposed architecture.

The trace file: An application specifies its requirements via a trace file. A trace file describes the *shape* of the throughput over time. It is composed of two-tuple entries in the form of *time* and *throughput*. An entry in the form $< i, T_i >$ indicates a request of T_i throughput at time instant i.

The preprocessor: The main goal of the preprocessor is to check the feasibility of the trace file and to create another feasible trace, if any of the checks fail. It performs two checks, a slope check and a region check. The slope check ensures that the requested throughput can be attained from one time instant to the next based on the round-trip time (RTT) of the Free connection. For any two successive time instants, i and j, the requested throughput T_j at time instant j, is bounded by:

$$T_j \leq T_i + \frac{j-i}{RTT^2} \quad (1)$$

Since TCP increases its congestion window by 1 packet every RTT, the congestion window at time j, cannot be more than $\frac{j-i}{RTT}$ of the congestion window at time i. Dividing by RTT to obtain the throughput leads to the above equation. The region check prevents the Stunts from achieving zero throughput. Thus for any time instant i, the requested throughput is bounded by:

$$T_j \leq s \times \bar{x}_s \quad (2)$$

where $\bar{x}_s$ is the average expected throughput per Stunt connection. This is a preliminary check and a more strict check is enforced online.

The controller: The controller decides which losses are picked up by the Free connection versus those delegated to the Stunts. The decision is based on the error signal between the current throughput and the requested throughput. We have experimented with two difference controllers. An On/Off controller and a Proportional Integral (PI) controller. An On/Off controller, decides the delegation percentage g_i, at time i according to the following equation:

$$g_i = \begin{cases} 0 & x_i > 1.3\bar{3}T_i \\ 1 & otherwise \end{cases} \quad (3)$$

where x_i is the instantaneous throughput of the Free connection at time instant i. An On/Off controller will try to delegate all the losses to the Stunts, whenever the current throughput is not matching the requested throughput. Otherwise, the Free connection will pick up its own losses and react to them. Notice that we compare x_i to $1.3\bar{3}T_i$ as opposed to T_i directly. This is due to the AIMD mechanism and the operation of the controller at short time-scales. A packet loss at $1.3\bar{3}T_i$ will cause the throughput to drop to $0.6\bar{6}$, leading to the correct average value of T_i, assuming that the RTT is kept constant.

The above controller may lead to oscillations, thus we experiment with a PI controller that adjusts the delegation percentage based on the following equation:

$$g_i = g_{i-1} + K \times (x_i - 1.3\bar{3}T_i) \quad (4)$$

where K is a constant the decides the aggressiveness of the controller in reaction to the error signal between the

current throughput and the requested throughput. The higher the value of K is, the more aggressive the controller would react.

B. Delegation of Congestion Signals

The Free and the Stunts architecture capitalizes on the fact that a delegation of a congestion signal from the Free connection to the Stunts will both (1) allow the Free connection to achieve a higher target data rate, and (2) it will not violate *any* TCP congestion control rules.

Since the Free connection can delegate congestion signals, it can continue sending as dictated by the Additive Increase component of the AIMD mechanism, by increasing its congestion window by 1 packet every RTT. Since the requested throughput is slope-checked by the preprocessor, then it should be able to achieve the requested target. However, in some conditions (explained below), the Free connection may not be able to delegate a congestion signal and thus it would have to cut its congestion window in half.

It is very important to realize that the impact of a congestion signal is not the same – as far as the network is concerned – whenever it gets delegated, since one connection may have a different congestion window size than the other. Thus, the reduction in the sending rate will not be equivalent.

In our particular case, to make sure that the network sees an equivalent reduction, we first check to see if the total congestion windows (of all the Stunts) is larger than that of the Free connection. If so, then we go through the Stunts, one by one in a round robin fashion, and we halve each one's congestion window, until the total reduction is at least as large as half the Free connection's congestion window size. If not, then we cannot delegate this loss and the Free connection has to react to it in the normal way, since we can only cause each Stunt connection to back-off one time for a given loss. Note that the round robin algorithm may cause the last Stunt connection to decrease its rate by a bit more of what is actually required, since we do not optimize to find the best fit among the Stunt connection's congestion window sizes that would sum to exactly the Free connection's congestion window size. This is wasted bandwidth that is essentially given up by the Stunts to be acquired by all connections. The effect of this is diminished over time as all connections grab more throughput.

The reason we go in round robin fashion is to provide some notion of fairness across the Stunts without hurting any one or group of Stunt connection. Notice that TCP fairness in our case is considered globally across the group of Free and the Stunt connections, as they can be abstractly considered as a single entity. The group of Free and Stunt connections should not together be more aggressive than an equivalent number of ordinary TCP flows when increasing their data flow rate through the normal TCP rules.

Paying back the Stunts: In some situations, the Free connection may not need a higher data rate. Either because the requested throughput at some point in time may go under its fair share or its data rate has increased to a point that is above the requested target rate. In both cases the Free connection can easily give up this bandwidth by having the application send less data. However, this slack of bandwidth will be naturally acquired by all connections (Stunts and cross-traffic). We have modified both controllers described above to allow for the reverse loss delegation from the Stunts to the Free connection. Reverse delegation helps the group of Free and Stunt flows to retain bandwidth as a whole, as apposed to releasing it to the network. Furthermore, it allows the Free connection to closely match its target when the target is low (typically below its fair-share). Also, as discussed above, delegation in this case would ensure that the Free connection would have a larger congestion window than the Stunt that is delegating. Otherwise, the Stunt connection cannot delegate a loss and would have to react to it.

We have chosen to delegate losses during the AIMD behavior, since we focus in this paper on longer data transfers with TCP. It is possible to delegate other behaviors such as timeouts and slow-start, but we do not consider those in this work for reasons having to do mostly with complexity and rareness of those particular events, in comparison to the AIMD behavior, on a well provisioned network.

IV. The Model

We extended a nonlinear fluid model, similar to those proposed in [21], [22], [23], [24], to capture the performance of m TCP flows traversing a bottleneck of capacity C, where m is equal to $(1 + s + n)$ as depicted in Figure 1.

A. Model Derivations

The round trip time $r_i(t)$ at time t for connection i is equal to the round-trip propagation delay D_i between the sender and the receiver for connection i, plus the queuing delay at the bottleneck router. Thus $r_i(t)$ can be expressed by:

$$r_i(t) \quad = \quad D_i + \frac{b(t)}{C} \tag{5}$$

where $b(t)$ is the backlog buffer size at time t at the bottleneck router. We denote the propagation delay from sender i to the bottleneck by D_{s_ib}, which is a fraction α_i of the total propagation delay.

$$D_{s_ib} \quad = \quad \alpha_i D_i \tag{6}$$

The backlog buffer $b(t)$ evolves according to the equation:

$$\dot{b}(t) \quad = \quad \sum_{i=1}^{m} x_i(t - D_{s_ib}) - C \tag{7}$$

which is equal to the input rate $x_i(.)$ from the m connections minus the output link rate. Notice that the input rates are delayed by the propagation delay from the senders to the bottleneck D_{s_ib}.

We assume RED, as proposed in [25], is employed at the bottleneck link as an active queue management scheme. Thus, the congestion loss probability $p_c(t)$ is given by:

$$p_c(t) = \begin{cases} 0 & v(t) \leq B_{min} \\ \sigma(v(t) - \varsigma) & B_{min} < v(t) < B_{max} \\ 1 & v(t) \geq B_{max} \end{cases} \quad (8)$$

where σ and ς are the RED parameters given by $\frac{P_{max}}{B_{max}-B_{min}}$ and B_{min}, respectively, and $v(t)$ is the average queue size, which evolves according to the equation:

$$\dot{v}(t) = -\beta C(v(t) - b(t)), \qquad 0 < \beta < 1 \quad (9)$$

Notice that in the above relationship, we multiply β by C since RED updates the average queue length at every packet arrival, whereas our model is a fluid model as indicated in [21], [23].

The loss delegation between the Free and the Stunts causes them to pick up different congestion signals than those set by RED. In particular, the Free connection, upon delegating $g(t)$ of its congestion signals, would pick up:

$$q(t) = p_c(t) - g(t) \quad (10)$$

Each Stunt connection would pick up:

$$q(t) = p_c(t) + \frac{g(t)}{s} \quad (11)$$

The normal cross-traffic are not affected and will simply pick up:

$$q(t) = p_c(t) \quad (12)$$

The throughput of TCP, $x_i(t)$ is given by

$$x_i(t) = \frac{w_i(t)}{r_i(t)} \quad (13)$$

where $w_i(t)$ is the size of the TCP congestion window for sender i .

According to the TCP Additive-Increase Multiplicative-Decrease (AIMD) rule, the dynamics of TCP throughput for each of the m connections can be described by the following differential equations:

$$\begin{aligned} \dot{x}_i(t) &= \frac{x_i(t - r_i(t))}{r_i^2(t)x_i(t)}(1 - q(t - D_{bs_i}(t))) \\ &- \frac{x_i(t)x_i(t - r_i(t))}{2}(q(t - D_{bs_i}(t))) \\ i &= 1, 2, .., m \end{aligned} \quad (14)$$

where $q(.)$ is the congestion signals observed by each connection based on its type. The first term represents the additive increase rule, whereas the second term represents the multiplicative decrease rule. Both sides are multiplied by the rate of the acknowledgments coming back due to the last window of packets $x_i(t - r_i(t))$. In the above equations, the time delay from the bottleneck to sender i, passing through the receiver i, is given by

$$D_{bs_i}(t) = r_i(t) - D_{s_ib} \quad (15)$$

Mode Assumptions: The model above makes the following assumptions: (1) It ignores the effect of slow-start and timeout mechanisms of TCP, since our main focus is on the AIMD. (2) The delegation of some losses can be distributed in a linear fashion among the Stunts (as indicated in Equation 11). In general, this does not hold except for small value of losses, since the throughput is inversely proportional to the square-root of the loss probability. Despite these assumptions, however, the model above still captures the main dynamics as we illustrate below.

B. Numerical Results

We instantiate the model above with specific parameters and we solve it iteratively. We assume there is 1 Free connection, 4 Stunts and 15 cross-traffic, for a total of 20 connections. The bottleneck has a capacity 2000 packets/sec. The RTT for each connection is chosen at random around 100 msec.

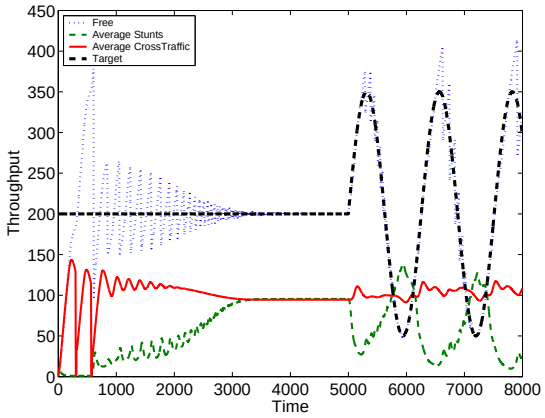

Figure 3. Numerical Results.

Figure 3 illustrates the performance of the Free connection in matching a target trace that starts with constant throughput at 200 packets/sec and then follows a sin wave. The figure also shows the average throughput across the stunts as well as the average throughput across the cross-traffic connections. One can observe how the Stunt connections make room for the Free connection to match the target throughput. Notice also, how little the normal cross-traffic is affected, except for the initial startup time (the first 3 seconds) where the whole system is still in a transient behavior. One can also see the impact of reverse delegation around time 6000.

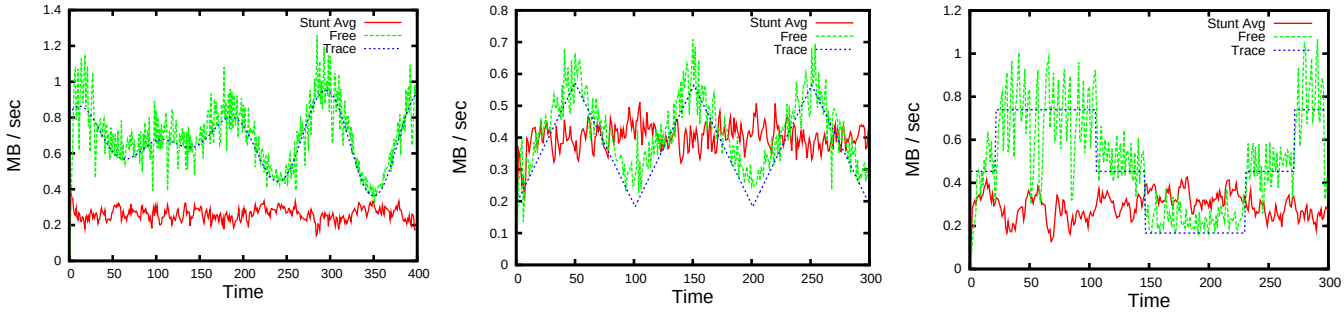

Figure 4. Three simulation traces to assess the Free connection's throughput in matching the target throughput.

Since the target throughput drops below the fair-share (100 packets/sec), the Stunts can delegate congestion signals to the Free connection and thus they are able to increase their throughput a bit above their fair-share.

V. Simulation Experiments

We have implemented our proposed architecture in NS-2 [26]. In this section, we study the performance of our proposed architecture under differing environments (topologies and trace files) and parameters.

The Setup: Figure 1 depicts the general topology of the simulated network. It is composed of a single bottleneck link that is traversed by the Free, Stunts, and cross-traffic connections. We assume all connections have an infinite supply of data to transmit. However, to study the impact of different dynamics that arise in practice, a number of the cross-traffic connections are turned on and off at varying times during a given simulation run. We also vary the number of Stunt connections to demonstrate and examine the behavior of our proposed architecture under different congestion levels.

The bottleneck link is configured with RED [25]. The queue size at the bottleneck link is chosen to be $\frac{RTT \times C}{\sqrt{m}}$ as advocated in [27], where C is the bottleneck link capacity and m is the total number of connections traversing the bottleneck. This is because on fully utilized simulation networks, those composed of only long lived TCP flows, the justification for dividing by the square root of m breaks down due to synchronization of network flows [27] [28]; however, when randomized cross traffic flows are added the premise is regained for the reduction in the buffer size. The RED parameter B_{min} is set to 0.25 the size of the queue resulting in the distance between B_{min} and B_{max} being three times B_{min}. Other parameters were chosen to encourage the stability of the average queue size.

Performance Metrics: To measure the effectiveness of our proposed architecture in matching the achievable throughput to the requested throughput, we propose a weighted variant of the standard "sum-of-squared errors" method. The main problem with the standard "sum-of squared errors" is that it does not differentiate between the case where the achieved throughput is above the target, versus the case where the achieved throughput is below the target (since in both cases we may get the same value). So we define the positive variance V^+ to be:

$$V^+ = \frac{\sum(x_i - T_i)^2}{C^+} \quad \forall \; x_i > T_i \tag{16}$$

where C^+ is the number of times (sample points) the achieved throughput is above target. Similarly, we define the negative variance V^- to be:

$$V^- = \frac{\sum(x_i - T_i)^2}{C^-} \quad \forall \; x_i < T_i \tag{17}$$

where C^- is the number of times (sample points) the achieved throughput is below target. To capture the overall performance we use a weighted variance, defined by:

$$V^* = \delta \times \frac{\sum(x_i - T_i)^2}{C^+ + C^-} \tag{18}$$

where δ is a ratio that is given by:

$$\delta = \frac{max(V^+, V^-)}{min(V^+, V^-)} \tag{19}$$

where δ is always greater than or equal to 1. If the matching is achieved ideally, then δ would be 1. A larger value of δ indicates a bias in the matching, either above or below the target, and this would increase the weighted variance in turn. The above metrics are computed over an entire simulation experiment.

A. Matching the Target Throughput

This set of experiments assess the ability of the Free connection in matching its throughput to different target trace files.

Figure 4 shows representative results using three different trace files with three different parameters. All results were obtained using a PI controller and with 10 Stunt connections. In each one, we plot the target trace, the throughput of the Free connection and the average throughput across all Stunts. Figure 4 (left) was obtained using a topology with 40 Mbps

bottleneck link capacity. In order to provide some variability, 8 of the cross-traffic connections through the bottleneck were randomized at ten second on/off intervals with the exception of 2 cross-traffic flows, which were continuous. This experimentation ensured that the flows through the bottleneck did not experience many timeouts.

Figure 4 (middle and right) were obtained using a topology with 80 Mbps bottleneck link capacity. The number of cross-traffic links was kept constant at 20 connections. Overall, one can see that the Free connection does a fairly good job in matching the target throughput while the throughput achieved by the Stunts changes in tandem. Notice the larger oscillations at higher data rates; these are expected due to the normal behavior of the AIMD mechanism. We see the opposite effect at lower data rates, due to a smaller decrease in bandwidth.

Notice also that the reverse delegation of losses from the Stunts to the Free connection allows the Stunts to achieve higher rates than they would have achieved otherwise. The slack of bandwidth given up by the Free connection goes directly to the Stunts as opposed to going to the Stunts and the cross-traffic. This is evident from Figure 4 (right) from time 150 until around 235 seconds. During this time interval, the Stunts are achieving higher throughput than their fair share, since the Free connection does not need that throughput. This also confirms our numerical results in Figure 3.

This experiment makes it clear that the Free flow can acquire a variety of target waveforms. Experimentation showed that the limiting factors were virtually all related to network latency and congestion. We understand this to be due to the fact that the architecture is designed around TCP congestion signals. Naturally, the ability of the Free flow to achieve a requested target rate is dependent on the ability of the network to support that link utilization. As was mentioned earlier, Figure 4 (left) is an example of the Free flow simulated in a well provisioned network with moderate cross-traffic dynamics and link utilization. That figure also shows very good fitness with regard to the target waveform.

B. Impact of the Number of Stunt Connections

To study the impact of the number of Stunt connections on the performance of the Free connection, we vary the number of Stunts while holding all other parameters constant and we plot the weighted variance (as given in Equation 18) versus the number of Stunts. As mentioned in Section I, these Stunt connections already exist due to the normal operation of the server. We do not advocate creating them to make this architecture work.

Figure 5 shows the results obtained (the non-random cross-traffic plot), where each point represent an independent simulation run. One can observe that there is an optimal number of Stunts (around 5 or 6) that minimizes the

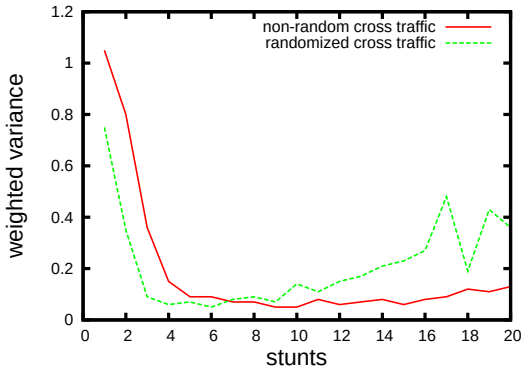

Figure 5. Impact of the number of Stunts on the weighted variance for non-random cross traffic and randomized cross-traffic.

weighted variance. Increasing the number of Stunts further, not only shows diminishing returns but also harms the performance of the Free connection as indicated by a slight increase in the weighted variance towards the higher number of Stunts.

Figure 6 shows the exact performances with 2, 10 and 20 Stunts, respectively. These plots were generated on topology of an 80 Mbps bottleneck link with 20 cross-traffic connections. Clearly, a very small number of Stunts (2) has a noticeable degrading effect on the performance of the Free connection, which improves with an increased number of Stunts up to a point where is starts decreasing again.

The number of Stunts affects the overall efficiency of this method because Stunts act more than just being a reservoir of bandwidth for the Free connection. In particular, they adjust the level of congestion in the network for the Free connection to better match the target throughput. If their number is very low, the Free connection would not be able to delegate losses (since we strictly enforce the same reduction in congestion windows from all Stunts). If their number is very large, the network would be more congested and all flows would go into timeouts/slow-start, which may prevent the Free connection to match the target throughput. One approach to handle this case would be to delegate timeouts, however, our focus in this paper was mainly on the AIMD mechanism as explained earlier.

C. Impact of Cross-traffic Dynamics

To study the impact of dynamics that arise in practice, we allow a number of the cross-traffic connections to be turned on and off at random times during a given simulation run. The cross-traffic flows are turned on and off every 10 seconds randomized with a uniform distribution.

Figure 7 shows the impact of varying the number of the cross-traffic connections, while keeping the number of Stunts steady at 10. We plot the positive variance, the negative variance, and the weighted variance to better explain a unique behavior observed in this experiment.

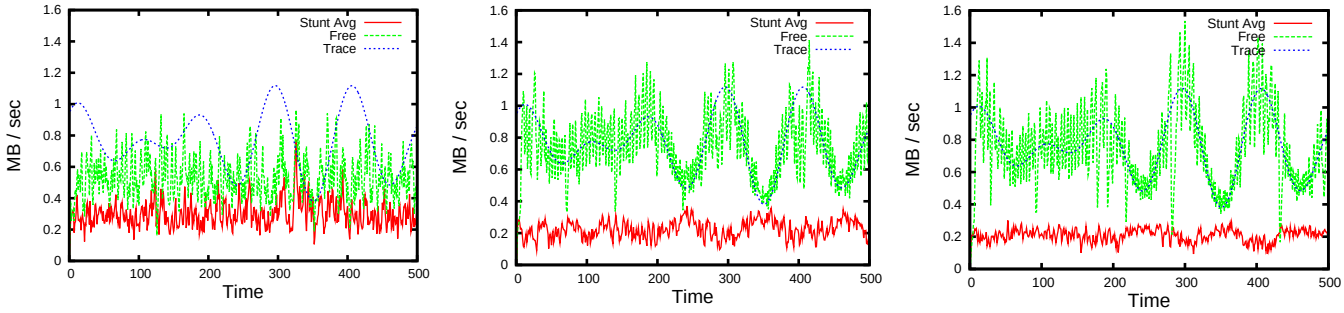

Figure 6. Impact of the number of Stunt connections on the performance. Left plot with 2 Stunts, middle plot with 10 Stunts and right plot with 20 Stunts.

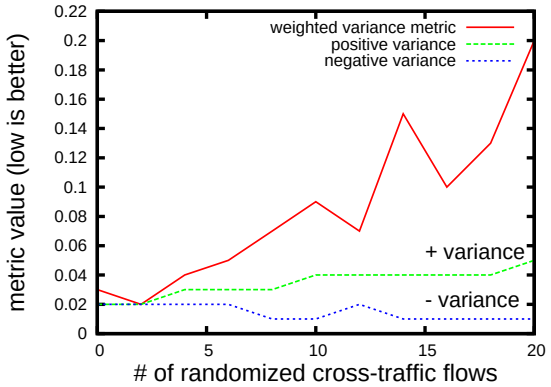

Figure 7. Impact of increasing the level of dynamic cross-traffic connections.

It is clear that the higher the dynamics from the randomized cross-traffic, the harder it is for the Free connection to match the target. However, when we examined the exact performance of the Free connection, we found that its shape was rather intact than deformed, but it was above the requested target. This was confirmed visually in each simulation run and is easy to see by examining the positive and negative variance metrics. Notice how the positive variance grows larger as the negative variance grows smaller. Such divergence increases the weighted variance due to a higher δ. The throughput of the Free connection was above the target because with a larger number of randomized cross-traffic, their combined throughput decreased the utilization at the bottleneck. This caused the Free connection to acquire more than the target because the lower link utilization also resulted in a lower packet loss probability.

Figure 5 (the randomized cross-traffic plot) shows the impact of the number of Stunts when most of the cross-traffic connections (19 out 20) were randomized on a 10-second on/off intervals. The presence of dynamics, coupled by an increase in the Stunts lead to a degraded matching between the the throughput of the Free connection and the target throughput.

D. On the Feasibility of the Free and the Stunts Architecture

As hinted in Section I, Internet measurement studies have indicated that around 80% of the traffic is controlled by a small number of connections (elephants) while the majority of connections (mice) control only around 20% of the traffic [4], [5], [6]. This is a direct effect of the heavy-tailed nature of file sizes on the Internet.

To study the execution of our proposed architecture in an environment with such traffic properties, we used a simulation network that is composed of 1 Free flow and 3 Stunt flows, all are elephant flows. In addition, the cross-traffic connections consisted of 4 elephant flows and 24 mice flows. Although, we initially classified flows as being elephants versus mice, such classification can be achieved dynamically as indicated in [29]. The bottleneck link is 10 Mbit with 10 ms latency. All links into and out of the bottleneck are 100 Mbit with 2 ms latencies. All the flows have an on/off state with a Pareto distribution with shape 1.5. The elephants have a mean burst time of 120 seconds and idle time of 5 seconds. The mice have a mean burst time of 1 second and idle time of 5 seconds. The parameters were chosen to produce a close distribution of traffic between elephants and mice that is close to the 80%-20% rule observed on the Internet. We limit the choice of Stunts to elephant flows.

Figure 8 (left) shows the results of an experiment run using the above parameters. We observe that, even with the presence of dynamics across all flows, utilizing the elephants as Stunts achieves a good matching between the Free connection's throughput and the target trace. That is because they control a large percentage of the capacity and are able to accept congestion signals delegations from the Free connection. Figure 8 (right) shows the results of a different experiment where the Free flow is turned on and off (we show the trace only in the on period of the Free flow). The elephant flows have a mean burst time of 10 seconds and idle time of 5 seconds. The mice flows have a mean burst time of 300 msec and 13 seconds. In this experiment, there were 10 elephant connections (only 4 were used as Stunts) and 50 mice connections. Again the parameters were chosen to produce the 80%-20% rule between elephant and mice. Utilizing few elephant flows as Stunts results in a good matching.

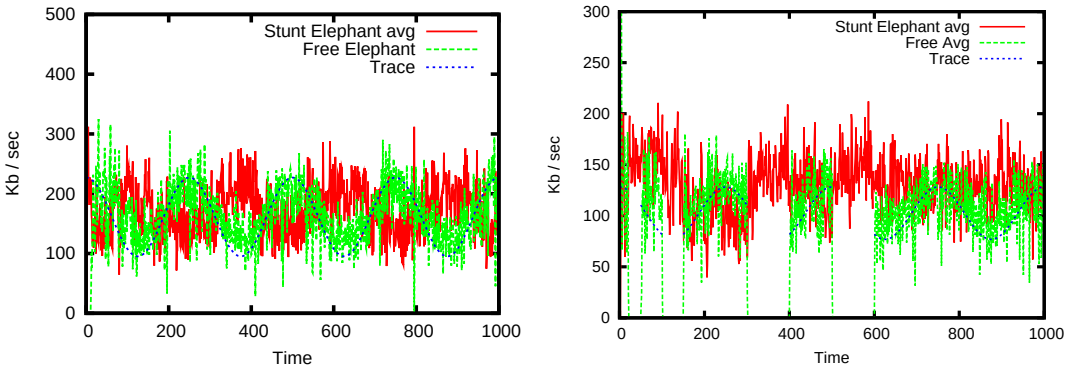

Figure 8. Simulation results with Pareto distribution. Elephants and mice consume around 80% and 20% of the bottleneck's capacity, respectively. Right plot considers the Free connection turning on and off as well.

VI. Implementation Experiments

We have implemented the Free and the Stunts architecture as a pluggable module for the 2.6.20 Linux kernel. We have chosen to experiment with the On/Off controller since the Linux kernel does not natively support floating point operations (implementing the PI controller is harder since it requires changing all math operations to fixed-point ones).

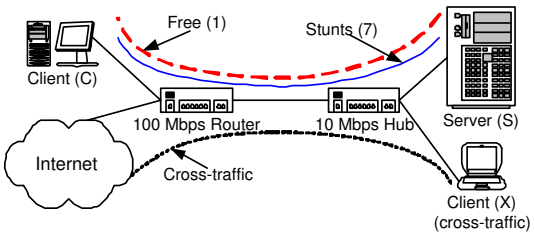

Figure 9. The experimental setup used in our implementation experiments.

The Setup: Figure 9 shows the experimental setup used in our implementation experiments. It is composed of a Server (S) and two Clients (C and X). The Server runs Linux and implements the Free and Stunts pluggable module. The server runs an application that accepts TCP connections from clients and serves them continuous flows of data. We have created a bottleneck link of 10 Mbps at the Server's first hop. All other links are 100 Mbps. All experiments were composed of 1 Free connection and 7 Stunt connections, from Client (C) to the Server. To simulate cross-traffic on the bottleneck link, we have manually opened and closed connections between Client (X) and different Internet web-servers.

A. Matching the Target Throughput

Our first set of experiments illustrates the performance of the free connection in matching the requested target. Figure 10 shows two different traces. Each plot shows the average throughput over four independent runs. We also plot the requested target as well as the adjusted target (1.33 of the requested target). One can observe that on average there is a very close matching between the throughput and the requested target.

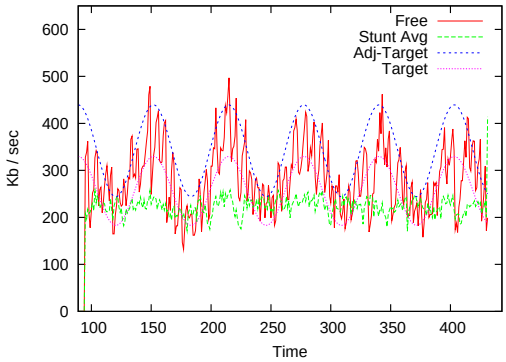

Figure 11. Free connection Performance with cross traffic introduced at time 330.

Figure 11 shows the performance of a single Free connection in matching the target throughput. One can see that the implementation does relatively well in matching the target. At around 330, we manually opened several connections from Client (X) to www.youtube.com over a period of 50 seconds and downloaded some videos in order to introduce cross traffic on the bottleneck link. One can see the effect of those cross-traffic connections on the performance on the Free connection as indicated by few misses in matching the target throughput.

B. Improving the Reverse Delegation

To improve the matching between the throughput of the Free connection and the target rate, we have modified the "reverse" delegation so that the Free connection does not drop its congestion window more than necessary. Recall that delegating a loss from a stunt connection to the free connection would cause the free connection to drop is congestion window by half. Here we modified such adjustment so that

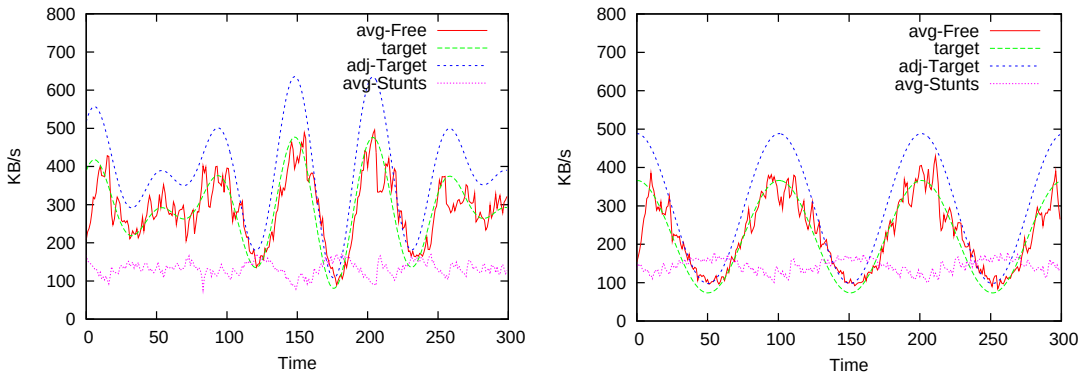

Figure 10. Implementation results for target matching.

the free connection would drop its congestion window by *the exact value* the delegating stunt would have experienced, if it were to pick up this congestion signal.

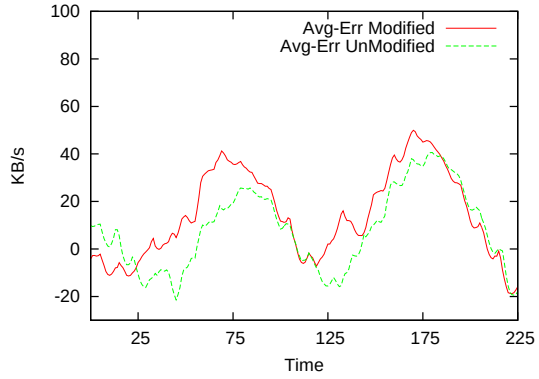

Figure 12. Improving the reverse delegation.

Figure 12 depicts the trending behaviors of the modified reverse delegation method versus the usual method, on the error signal obtained as the difference between throughput achieved and the target. The plotted lines are the average error values over several runs.

One can observe from Figure 12 that the trending of the targeting error is higher in certain areas of the plot. These areas are predictably located where reverse delegation has a higher probability of occurring (for example, the throughput of the Free flow needs to be reduced to meet the target). The increased error means that the Free flows throughput is trending closer to the adjusted target, rather than the requested target. We have experimented with different target traces and they show the same types of trends. This behavior was expected since the 1.33 target adjustment was computed based on the exact halving of the congestion window, due to normal TCP Congestion Control AIMD behavior. One can mitigate this effect by having a closer adjusted target to the requested target.

VII. Conclusion

This paper describes an architecture that enables a set of clients to delegate and trade congestion signals between them in order to shape traffic based on demands from the applications. The architecture strictly adheres to TCP rules and does not affect network congestion on the bottleneck links. Moreover, no modifications is required to any devices along the communication path (unless an ISP decides to implement this architecture to manage flows). We have shown that this service architecture is capable of providing a reasonably accurate targeting between the achieved rate and the target rate with a small number of Stunts. We have assessed the performance of our proposed architecture through new metrics, using numerical solutions, extensive simulation experiments and real implementation in the Linux kernel.

References

[1] J. Valdez and M. Guirguis, “Liberating TCP: The Free and the Stunts,” in *Proceedings of the 7th International Conference on Networking (ICN 2008)*, Cancun, Mexico, April 2008.

[2] V. Cerf and L. Kahn, “A Protocol for Packet Network Interconnections,” *IEEE Transactions on Communications*, vol. 22, no. 5, June 1974.

[3] V. Jacobson, “Congestion Avoidance and Control,” in *Proceedings of ACM SIGCOMM*, Stanford, CA, August 1988.

[4] K. Thompson, G. Miller, and R. Wilder, “Wide-Area Internet Traffic Patterns and Characteristics,” *IEEE Networks*, vol. 11, no. 6, 1997.

[5] S. Fred, T. Bonald, A. Proutiere, G. Régnié, and J. Roberts, “Statistical Bandwidth Sharing: A Study of Congestion at Flow Level,” in *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[6] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. Diot, “Packet-level Traffic Measurements from the Sprint IP Backbone,” *Network, IEEE*, vol. 17, no. 6, pp. 6–16, 2003.

[7] L. Guo and I. Matta, "The War between Mice and Elephants," in *Proceedings of IEEE ICNP*, Mission Inn, Riverside, November 2001.

[8] C. Barakat and E. Altman, "Performance of Short TCP Transfers," *Lecture Notes in Computer Science*, pp. 567–579, 2000.

[9] M. Mellia, H. Zhang, and D. di Elettronica, "TCP Model for Short Lived Flows," *IEEE Communications Letters*, vol. 6, no. 2, pp. 85–87, 2002.

[10] J. Londono, A. Bestavros, and N. Laoutaris, "Trade and Cap: A Customer-Managed, Market-Based System for Trading Bandwidth Allowances at a Shared Link," *Technical Report BUCS-TR-2009-025, CS Department, Boston University*, 2009.

[11] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," *RFC 1633, 1994*, June 1994.

[12] S. Blake, D. Black, M. Carlson, E. Davies, Z.Wang, and W. Weiss, "An Architecture for Differentiated Services," *IETF RFC 2475, 1998*, December 1998.

[13] H. Balakrishnan, H. Rahul, and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," in *Proceedings of ACM SIGCOMM*, Cambridge, MA, August 1999.

[14] D. Ott and K. Mayer-Patel, "An Open Architecture for Transport-level Protocol Coordination in Distributed Multimedia Applications," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), 2007*, August 2007.

[15] Z. Rosberga, J. Matthewsa, and M. Zukerman, "A Network Rate Management Protocol with TCP Congestion Control and Fairness for All," *Elsevier Computer Networks*, vol. 54, no. 9, June 2010.

[16] Z. Rosberg, J. Matthews, C. Russell, and S. Dealy, "Fair and End-to-End QoS Controlled Internet," in *Proceedings of 3rd International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ)*, Athens, Greece, June 2010.

[17] M. Guirguis, A. Bestavros, I. Matta, N. Riga, G. Diamant, and Y. Zhang, "Providing Soft Bandwidth Guarantees Using Elastic TCP-based Tunnels," in *Proceedings of the 9th IEEE Symposium on Computer and Communications*, Alexandria, Egypt, July 2004.

[18] T. Tsugawa, N. Fujita, T. Hama, H. Shimonishi, and T. Murase, "TCP-AFEC: An Adaptive FEC Code Control for End-to-End Bandwidth Guarantee," in *Proceedings of 16th International Packet Video Workshop*, Lausanne, Switzerland, November 2007.

[19] P. Siripongwutikorn, S. Banerjee, and D. Tipper, "A Survey of Adaptive Bandwidth Control Algorithms," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 1, pp. 14–26, 2009.

[20] H. Wei and Y. Lin, "A Survey and Measurement-based Comparison of Bandwidth Management Techniques," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 2, 2009.

[21] C. Hollot, V. Misra, D. Towsley, and W. Gong, "A Control Theoretic Analysis of RED," in *Proceedings of IEEE INFOCOM 2001*, Anchorage, AL, April 2001.

[22] F. Kelly, "Mathematical Modelling of the Internet," *Mathematics Unlimited and Beyond, 2001*, pp. 685–702, 2001.

[23] S. Low, F. Paganini, J. Wang, S. Adlakha, and J. Doyle, "Dynamics of TCP/RED and a Scalable Control," in *Proceedings of IEEE INFOCOM*, New York, NY, June 2002.

[24] S. Shenker, "A Theoretical Analysis of Feedback Flow Control," in *Proceedings of ACM SIGCOMM*, Philadelphia, PA, September 1990.

[25] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *Transactions on Networking*, vol. 1(4), pp. 397–413, August 1993.

[26] E. Amir and et al., "UCB/LBNL/VINT Network Simulator - ns (version 2)," available at http://www.isi.edu/nsnam/ns/.

[27] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proceedings of ACM SIGCOMM*, Portland, Oregon, August 2004.

[28] L. Zhang, S. Shenker, and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," in *Proceedings of ACM SIGCOMM*, Zurich, Switzerland, September 1991.

[29] K. Avrachenkovt, U. Ayesta, P. Brown, E. Nyberg, and F. INRIA, "Differentiation between Short and Long TCP Flows: Predictability of the Response Time," in *Proceedings of IEEE INFOCOM*, vol. 2, 2004.