Overlay Bandwidth Services: Basic Framework and an Edge-to-Edge Closed-Loop Building Block[1]

David Harrison, Shivkumar Kalyanaraman, Sthanunathan Ramakrishnan
Department of ECSE, Department of Computer Science,
Rensselaer Polytechnic Institute.
harrisod@cs.rpi.edu, shivkuma@ecse.rpi.edu, sthanu@networks.ecse.rpi.edu

1 Introduction

QoS deployment in multi-domain, IP-based inter-networks has been an elusive goal partly due to complex deployment issues [18]. This paper proposes a new *overlay* framework to support a limited range of bandwidth services while being attractive from a deployment standpoint (see Figure 1). The framework does not place *implementation or upgrade requirements* at bottlenecks; requires *no new packet-format reqts* at the IP-level; and is *incrementally deployable*. It does require *aggregate-level* isolation, buffers and bandwidth *configuration* at potential bottlenecks (i.e. one queue for *all* overlay traffic) which can be satisfied minimally by static methods. A new *closed-loop building block* is proposed to efficiently realize the overlay functionality (Eg: between nodes I and E in Figure 1). The building block assumes internal FIFO queues, rate-based control and proposes a new per-loop accumulation-based technique to transparently detect congestion without inducing packet-loss. In O(RTT) timescales, the queues are distributed to the edge nodes. We present proofs of multi-bottleneck stability/fairness and develop measurement procedures for this building block. At the edges a toolkit of stateful and application-aware policies can customize services for end-to-end applications [26, 22]. A medium scale site-to-site VPN is a sample target of this model.

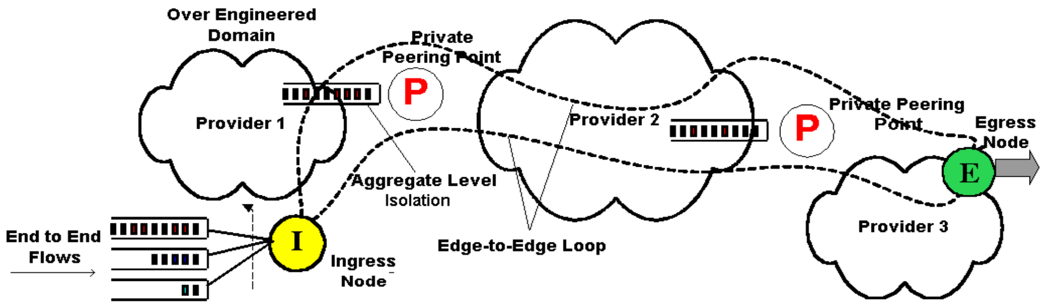


Figure 1: Overlay Bandwidth Services Model

A key tradeoff is that this framework *cannot offer tight delay or jitter guarantees.* The focus is on

[1]Submitted for conference review. Please do not re-distribute.

transparent edge-to-edge loss minimization and bandwidth assurances. Another tradeoff is that it requires fairly large buffers for *fully* lossless operation and assumes relatively stable edge-to-edge paths. However, this framework is *not inconsistent and can co-exist* with frameworks like int-serv, diff-serv, MPLS, CSFQ [7, 5, 36] which can provide a wider range of services, but involve more coordinated deployment. The closed-loop building block *does not conflict* with deployment of TCP-ECN [31] (ECN marking is unaffected by the scheme) and new end-to-end congestion-control schemes [15, 4, 3]. The mapping of end-to-end flows over edge-to-edge loops is safe because the latter *positively affects* end-to-end performance (see section 7). The building block also provides a mechanism for *edge-based isolation of misbehaving flows* [14, 36, 34, 25]. While the basic scheme can operate assuming FIFO queues everywhere (given aggregate-level isolation), ECN and AQM schemes [31, 16, 26] deployed *at the edges* lead to significant performance gain (see section 7). In summary, the framework and closed-loop building block aims *to complement, and not substitute* the state-of-the-art.

Section 2 positions this work in the context of current QoS and congestion control literature. Sections 3, 4, 5 present the theoretical basis, stability and performance bounds of the scheme in terms of a fluid flow model. Section 6 develops the packet-based algorithm, measurement techniques and parameter settings. A sample of overlay service capabilities, illustrated with simulations are presented in Section 7. Finally, section 8 discusses applicability, limitations and future work within the framework.

2 Inter-network QoS and Congestion Control: The Middle Ground

Contemporary QoS research has recognized the need to *simplify and de-couple* building blocks to promote implementation and inter-network deployment. The int-serv and RTP work [7, 35] de-coupled applications evolution from network support evolution. RSVP de-coupled inter-network signaling from routing. MPLS [33] de-coupled forwarding mechanisms from the routing control plane, leading to traffic engineering capabilities [2]. The diffserv services [5, 20, 10] and core-stateless fair queuing (CSFQ) [36] further simplified core architecture and moved data-plane complexity to the "edges," and allowed a range of control-plane options [33, 2, 6, 13]. Extending this trend, our proposed overlay framework *eliminates* implementation and upgrade requirements at potential intermediate bottlenecks (Eg: at peering points P in Figure 1), and further de-couples "edge" evolution from "core" evolution.

We propose to efficiently realize the overlay capability using a new closed-loop building block. Given the growth of optical networking, raw bandwidth and traffic engineering options, our closed-loop block is useful *only if* the loop actually encompasses bottlenecks, and if over-engineering is impossible, costly or cannot assure zero buffer overflow. A multi-provider or an international environment may pose such challenges justifying the applicability of this framework (Figure 1).

The *closed-loop* nature of our building block differs fundamentally from traditional blocks like schedulers and shapers which are open-loop in nature [17]. The price we pay is O(RTT) adaptation and the lack of delay/jitter assurances. Closed-loop blocks for services have been proposed in recent literature

[12, 30, 1]. But, most of these proposals suggest end-to-end operation which again faces deployment hurdles. Also, some of these proposals are *packet loss-based* which excludes a range of customizable-loss services, and may not be suited for assured bandwidth services.

The proposed building block has a useful set of properties: *edge-to-edge overlay operation, stability, bounded buffer, lossless accumulation-based congestion detection, proportional fairness [22, 27, 23], measurement feasibility, parameter-setting feasibility and flexible policy options* which make it attractive as a building block for overlay services. In contrast, TCP- and TCP-friendly congestion control [19, 15, 4, 3, 14] operate *end-to-end* using packet loss as a congestion indicator, and may require network element support for performance enhancement [16, 26]. Our block differs from schemes like DECbit [32], TCP-ECN [31], ATM ABR explicit-rate control [9], hop-by-hop control [1, 24], credit-based schemes [24] and others which may require *varying degrees of computation, buffer management or fine-grained scheduling support at bottlenecks.* Our notion of "edge-to-edge overlay" is motivated by the diff-serv model [5], and therefore is *neither* "end-to-end" (like [19, 4, 8, 15]) *nor* "hop-by-hop" (like [1, 24]).

The property of detecting congestion without inducing packet loss was introduced in schemes like Vegas [8, 28], Agrawal/Cruz et al [1] and Mo/Walrand [29][2]. However, we propose a *new accumulation-based* technique for congestion detection. Assuming no packet loss or system delay changes, the per-loop accumulation is simply the backlog of that loop in the network, i.e., the difference between the packets sent and received for a given loop (see section 3). It directly correlates to the conservation-of-packets principle [19]. In *combination with ingress rate-control*, it allows the use of simple thresholding techniques to detect congestion; and leads to a proof of stability and fairness in a fluid-flow model. The distinction between "accumulation" and other congestion measures is given in Section 3, but its precise relationship with measures like "delay" under highly time-varying network conditions is yet to be fully understood.

To summarize, the two key contributions in this paper are: an overlay framework (QoS architectural contribution), and the development of an accumulation-based congestion detection technique to fit into an edge-to-edge *near*-lossless control scheme (congestion control contribution).

3 Fluid Model

This section introduces the concept of "accumulation" in a fluid flow model and motivates various elements of the building block. Consider a network of queues fed by a system of fluid flows { i } with input rates $\lambda_i(t)$ at the ingress, and output rates $\mu_i(t)$ at the egress (Figure 2). The circles in Figure 2 represent bottlenecks (i.e. capturing both the link and the queue). We assume infinite buffers at the bottlenecks. Let $\lambda_{ji}(t)$ be the input rate, and $\mu_{ji}(t)$ be the output rate of flow i w.r.t the j^{th} bottleneck at time t. Assume that this flow goes through J_i bottlenecks. For notational simplicity, we denote J_i

[2]Our work was concurrently performed with that of the latter authors[28, 1, 29]

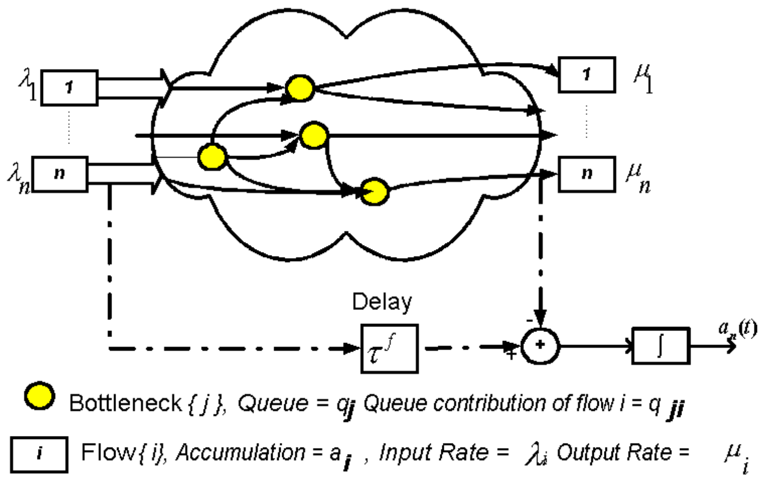


Figure 2: Fluid Flow Model of a Multi-Bottleneck Network with Out-of-Band Measurement

by J and j_i by j in what follows[3]. Let the delay between bottlenecks j and $j+1$ be d_j. Now the *input rate* of flow i at any bottleneck $j+1$, $\lambda_{(j+1)i}(t)$ is the time-shifted version of the *output rate* of the same flow at the *previous* bottleneck j, i.e.,$\lambda_{(j+1)i}(t) = \mu_{ji}(t-d_j)$. Note that $\lambda_i(t-d_0) = \lambda_{0i}(t)$, and $\mu_{Ji}(t-d_J) = \mu_i(t)$. The queue contribution of flow i at bottleneck j at time t, $q_{ji}(t)$ follows the dynamics:

$$\dot{q}_{ji}(t) = \lambda_{ji}(t) - \mu_{ji}(t)$$

Define the total accumulation of flow i, $a_i(t)$ by the dynamics:

$$\dot{a}_i(t) = \sum_j \dot{q}_{ji}(t - \sum_{r=j}^{J} d_r) = \lambda_i(t - \sum_{r=0}^{J} d_r) - \mu_i(t) \tag{1}$$

Here $\sum_{r=0}^{J} d_r = \tau_i^f$ is the forward propagation delay for flow i. Let τ_i^b be the reverse propagation delay. We assume *continuous out-of-band* signaling of rate-measures between ingress and egress. The meaning of continuous out-of-band is that control information is sent *continuously*, and received at the egress after a delay τ_i^f and returned to the ingress after a total delay $\tau_i^f + \tau_i^b$ (i.e. *out-of-band*). Now, assuming $a_i(0) = 0$, $a_i(t)$ can be calculated at the egress knowing $\lambda_i(t - \tau_i^f)$ and measuring $\mu_i(t)$ (as illustrated in Figure(2). Consider an interval $\tau > \tau_i^f + \tau_i^b$ for which the input rates $\lambda_i(t)$ are held constant, and that the intervals used by various flows are equal and synchronized. Then, the system may be thought of as progressing in cycles of length τ at *any* node (ingress, bottleneck or egress), albeit time-shifted. In other words, the output rate of a flow i from bottleneck j in its path is the same as its input rate to the next bottleneck $j+1$, i.e.: $\forall j \;\; \mu_{ji}(k) = \lambda_{(j+1)i}(k)$, and

$$a_i(k) = \sum_j q_{ji}(k) \tag{2}$$

[3]Eg: A term like $\lambda_{ji}(t)$ really means $\lambda_{j_i i}(t)$

Equation (2) shows that the accumulation measured $a_i(k)$ is not necessarily equal to the flow's contribution at any one bottleneck on its path $q_{ji}(k)$. Discretizing Equation (1), we obtain:

$$a_i(k) = a_i(k-1) + (\lambda_i(k) - \mu_i(k))\tau \tag{3}$$

In a *fluid model*, if $\lambda_{ji}(k) > 0 \;\; \forall\, i$, then for any positive queue at bottlenecks, there exists a positive queue contribution of every participating flow and vice versa. Moreover, the bottleneck queue is zero iff the queue contribution of every participating flow is zero. Therefore:

$$q_j(k) > 0 \Leftrightarrow \;\forall i \;\; q_{ji}(k) > 0, \text{ and } q_j(k) = 0 \Leftrightarrow \;\forall i \;\; q_{ji}(k) = 0 \tag{4}$$

From Equation (2) and (4), it follows that the accumulation measured by a fluid flow $a_i(k)$ is positive iff there is *at least* one bottleneck in its path with a non-zero queue; and $a_i(k) = 0$ iff all the bottlenecks in its path have a zero queue. That is,

$$a_i(k) > 0 \Leftrightarrow \;\exists j \;\; q_j(k) > 0, \text{ and } a_i(k) = 0 \Leftrightarrow \;\forall j \;\; q_j(k) = 0 \tag{5}$$

Therefore $a_i(k)$ is an indicator of congestion. In other words, with the fluid-model and continuous feedback, using a *zero-threshold for the entire queue at the bottleneck $q_j(k)$ is equivalent to using a zero-threshold for the accumulation $a_i(k)$*. Now we adopt the following policy for rate updation:

$$\lambda_i(k+1) = \lambda_i(k) + \alpha; \;\; a_i(k) = 0 \tag{6}$$

$$\lambda_i(k+1) = \min\{\lambda_i(k), \beta\mu_i(k)\}; \;\; a_i(k) > 0 \tag{7}$$

Equations (6) and (7) represent the core policy of the proposed closed-loop building block. The parameters α, β are constants ($\alpha > 0, \beta < 1$)affecting rate increase and decrease respectively. NETBLT [11] shares the property of measuring output rates $\mu_i(k)$, but it detects congestion based on the rate-differences $\lambda_i(k) - \mu_i(k)$, instead of the accumulation, $a_i(k) = \sum_k(\lambda_i(k) - \mu_i(k))T$. Our policy is peripherally similar to AIMD[19, 32], but uses the *output rate* $\mu_i(k)$ for rate-decrease during congestion epochs. Hence we call this policy as AIMD-ER (AIMD-Egress Rate). Accumulation ($\sum_j q_{ji}(t)$) is distinct from network delay ($\sum_j \frac{q_j(t)}{C_j(t)}$) used in schemes like Vegas, Mo/Walrand, Agrawal/Cruz [8, 28, 1, 29], especially when $C_j(t)$ varies. The precise relationship between these models under complex conditions is an open issue. We now examine the stability and fairness characteristics of the scheme.

4 Stability and Queue Bounds

Consider flow i in congestion between cycles $k = p$ and $k = n$. Since the flow is uncongested at $k = p-1$, $a_i(p-1) = 0$. From Equations (3) and (7), we have:

$$a_i(n) \;\; = \;\; \sum_{k=p}^{n}(\lambda_i(k) - \mu_i(k))\tau, \;\; \text{and} \tag{8}$$

$$\forall k > p: \quad \lambda_i(k) \quad \leq \quad \beta\mu_i(k-1) \tag{9}$$

Substituting Equation (9) in (8), and combining terms we obtain:

$$a_i(n) \leq (\lambda_i(p) - (1-\beta)\sum_{k=p}^{n-1} \mu_i(k) - \mu_i(n))\tau \tag{10}$$

Let us first prove stability for a *single bottleneck* j experiencing congestion from time $k = p$ to n. Denote the set of flows passing through the bottleneck by $\mathcal{N}_b$, with cardinality N_b. Since the bottleneck is uncongested at $k = p - 1$ and from Equations (6),(7),(2),(10), we have:

$$\sum_{i \in \mathcal{N}} \lambda_i(p-1) \leq C; \quad \sum_{i \in \mathcal{N}} \lambda_i(p) \leq C + N_b\alpha$$

$$\begin{aligned} q_j(n) = \sum_{i \in \mathcal{N}} a_i(n) \quad &\leq \quad \sum_{i \in \mathcal{N}} (\lambda_i(p) - (1-\beta)\sum_{k=p}^{n-1} \mu_i(k) - \mu_i(n))\tau & (11) \\ &\leq \quad (N_b\alpha - (1-\beta)(n-p)C)\tau & (12) \end{aligned}$$

Therefore, the single bottleneck case is stable with queue bound $N_b\alpha\tau$ and the queue drain time is $\frac{N_b\alpha\tau}{(1-\beta)C}$.

Now, let us consider a *general multi-bottleneck network* which was uncongested for $k < 0$. The queue at any *one bottleneck* j at time $n > 0$ is less than or equal to the sum of queues at *all bottlenecks* at time n. Flows can be classified into 3 sets:

CF-flows congested from $k = 0$ to n,

CF^e- flows congested from $k = 0$ and upto $k = m$ $(0 < m < n)$.

CF^b-flows congested starting from $k = p$ $(p > 0)$.

The above notation is simplified. We indeed assume that the time periods m and p are not necessarily the same for all flows in the sets. Also, the flows which are a part of CF^e may later become part of CF^b. The *sum of queues at all bottlenecks is the same as the sum of accumulation of all the flows.* Therefore:

$$q_j(n) \leq \sum_{i \in CF \cup CF^e \cup CF^b} \sum_{k=0}^{n} (\lambda_i(k) - \mu_i(k))\tau \tag{13}$$

The flows belonging to set CF^e do not add to the queue length at time n (follows from Equation (5)), and the sets CF and CF^b are disjoint. Therefore:

$$q_j(n) \quad \leq \quad \sum_{i \in CF} \sum_{k=0}^{n} (\lambda_i(k) - \mu_i(k))\tau + \sum_{i \in CF^b} \sum_{k=p_i}^{n} (\lambda_i(k) - \mu_i(k))\tau$$

Applying Inequality(10) to the RHS:

$$
\begin{aligned}
q_j \quad (n) &\leq \sum_{i \in CF} \left(\lambda_i(0)\tau - (1-\beta) \sum_{k=0}^{n-1} \mu_i(k)\tau - \mu_i(n)\tau \right) + \sum_{i \in CF^b} \left(\lambda_i(p_i)\tau - (1-\beta) \sum_{k=p_i}^{n-1} \mu_i(k)\tau - \mu_i(n)\tau \right) \\
&\leq \sum_{i \in CF} \lambda_i(0)\tau + \sum_{i \in CF^b} \lambda_i(p_i)\tau - (1-\beta)\tau \left(\sum_{i \in CF} \sum_{k=0}^{n-1} \mu_i(k) + \sum_{i \in CF^b} \sum_{k=p_i}^{n-1} \mu_i(k) \right) - \tau \left(\sum_{i \in CF} \mu_i(n) + \sum_{i \in CF^b} \mu_i(n) \right)
\end{aligned}
$$

The sum of input rates can be bounded by their access link capacities and hence $\sum_{i \in CF} \lambda_i(0) + \sum_{i \in CF^b} \lambda_i(p_i) \leq NC$, where N is the number of sources in the entire network. Note that this inequality is true irrespective of p_i and the relative distribution of flows between CF and CF^b. Also [4]$\sum_{i \in CF \cup CF^b} \mu_i(k) \geq C$. This is because when the flows are congested the sum of output rates at some bottleneck is equal to C, and there exists at least one bottleneck from which the output rates are directly measured at the egress. For example, Figure 3 shows three coupled bottlenecks. If bottleneck 3 is congested initially, flows 1 and 3 are congested and $\mu_1 + \mu_3 = C$. Suppose suddenly 1 and 2 become bottlenecks, and 3 becomes uncongested, then all three flows are congested and $\mu_1 + \mu_2 = C$ and hence $\mu_1 + \mu_2 + \mu_3 \geq C$. This example shows that as long as there is one congested flow this inequality is valid, irrespective of the bottleneck or the actual flow getting congested. This means that $\forall k \sum_{i \in CF} \mu_i(k) + \sum_{i \in CF^b} \mu_i(k) \geq C$ as long as the sets are not empty at time k.[5] When we plug this bound in Equation (14) above, we see that the queue bound decreases with n. So to obtain the worst case we have to assume that the sets CF and CF^b are empty for as large a k as possible. Clearly the largest such k has to be $n-1$ as at n there should be some congested flows to result in a queue. This means that the set CF should be empty and that all the flows belonging to CF^b should get into congestion at $k = n$. Incorporating all this we get,

$$
q_j(n) \quad \leq \quad (N-1)C\tau \tag{14}
$$

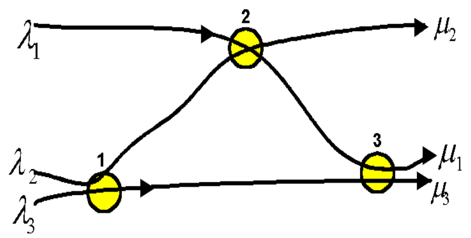


Figure 3: Coupled Bottlenecks: Lower Bound on Aggregate Egress Rate ($\sum_{CongestedFlows} \mu_i(k)$) is C

The last equation assumes equal capacities[6], C. The maximum queue is given by $q_{max} = (N-1)C\tau$, which proves system stability. However, the above bound is weak[7]: $O(NC\tau)$ where N is the number of

[4] If the capacities are different then $\sum_{i \in CF \cup CF^b} \mu_i(k) \geq \min_j \{C_j\}$

[5] Note that if $k < p_i$, then that i^{th} flow is not yet an element of CF^b. This confusion is the price we pay for the notational simplicity.

[6] If capacities are different then we replace NC by the sum of the link capacities

[7] Because our technique bounds any *single* queue by the *sum of all queues*, and assumes the maximum values for all λ_i.

flows in the *entire network*! The complex coupling between bottlenecks defies exact analysis. However, Section 5 shows that the system tends to proportional fairness. Therefore, we expect the sum of output rates of flows at any bottleneck to be bounded away from 0 by $\sum_i \min_k\{\mu_i(k)\}$ (see ref.[29]) under steady state conditions. Then, from Equation (11) we can derive $q_{max} = (N_bC - \sum_i \min_k\{\mu_i(k)\})\tau = O(N_bC\tau)$ since $\sum_{i\in\mathcal{N}_b}\lambda_i(p)$ is bounded by N_bC.

This analysis assumes a fixed and equal τ for flows. Consider a setting of τ_i (for flow i) larger than its round trip propagation delay instead of an equal τ for all flows. Now, accumulation a_i for any flow still measures the sum its queue contributions q_{ji} (from Equation (3), albeit at different times and the relations (4) hold at any instant. Therefore, accumulation is still a reliable indicator of congestion, but the feedback is effected by the ingress nodes at different times. Stability still holds because the proof depends only upon the fact that $\lambda_i(t) < \beta\mu_i(t-\tau_i)$ during congestion, and that $\sum_{flows}\lambda_i(t)$ is upper-bounded by NC and $\sum_{congestedflows}\mu_i(t)$ is lower-bounded by C throughout the congestion epoch.

5 Fairness

In this section we take the approach used in [23, 22]. Lets assume that there is a given probability P that a flow is congested, independent of the flow rate (holds for high degrees of multiplexing at bottlenecks). Then the input rate of a flow i in round $k+1$ is given by:

$$\lambda_i(k+1) = (1-P)(\lambda_i(k)+\alpha) + P\beta\mu_i(k) \tag{15}$$

Recall that $\tau > \tau_i^f + \tau_i^b$. Now approximating the finite difference as the derivative, we have:

$$\dot{\lambda}_i(t) = (1-P)\frac{\alpha}{\tau} - \frac{P}{\tau}(\lambda_i - \beta\mu_i) \tag{16}$$

Kunniyur and Srikant [23] proved that this policy leads to a game in which the optimization function for the entire network is given by:

$$\max_{\{\lambda_i\}} \sum_i \left(\frac{\alpha(1-P)}{\beta P}\right)\log\lambda_i - P\sum_{links}\int_0^{\sum\lambda_i} p_j(C,\lambda)d\lambda - P\frac{1-\beta}{\beta}\lambda \tag{17}$$

where λ is the vector of input rates, sum on the upper limit of the integral is over the the set of flows which pass through link j and $p_j(C,\lambda)$ is defined by $\sum_{links}\lambda_i p_j(C,\lambda) = \lambda_i - \mu_i$. If β is very close to 1 and if $\frac{P}{(1-P)\alpha}$ is large, then the scheme approaches proportional fairness [22, 27]. This can be done by setting α small. Also the weights for different sources in the utility functions can be modified by setting different values of α and β, thereby creating a *weighted service* to the flows. The proportional fairness property makes it similar to the window-based schemes in [29]. A future work item is to extend our scheme to achieve (p,α)-proportional[8] fairness [29] which tends to max-min fairness and is known to be *theoretically* feasible for window-based control.

[8] The α here should not be confused with the rate-increase parameter

6 Measurement and Parameter Issues

Our task is now to extend the fluid-flow model to account for packetized transmission, and *in-band* control-flow between the ingress/egress. For simplicity of exposition, lets consider equal-sized packets. The building blocks can be modified appropriately to support variable-size packets. The input rates $\lambda_i(k)$ are assumed to be regulated by token buckets with maximum burst size $\sigma = 1$ packet. The first problem that occurs is that at any queueing point, the queue *even without overload* can be upto $n\sigma$ at any bottleneck,where n is the number of flows through the bottleneck. This leads to randomness in the accumulation measure, $a_i(k)$. Recall that a fluid-flow could not have a queue during underload which allowed a congestion detect threshold of zero, i.e., $a_i(k) > 0$ signalled congestion. The mean number of packets that are received in a time interval at the egress node is equal to that transmitted in the same time interval (because the mean difference of queuing delays between packets is zero). The number of packets can change only if a packet from outside the interval is received in this interval and vice versa. Without a large degree of error, we can assume that the probability of two or more packets moving out of an interval is very less. If p is the probability that a packet goes out of its interval, it can be shown that the variance is $2p(1-p)$ which is upperbounded by 0.5. Also the average error in counting of packets at the ingress and egress could be as much as 0.5 packets. To reduce the probability of false congestion detections, we set the epoch-beginning threshold (H_{thresh}) to 2 pkts, and the end-of-epoch threshold (L_{thresh}) at 1.5 pkt. This setting is robust, but affects the average and worst case queue length.

6.1 Measurement of Accumulation

The critical element of the building block is the concept of "accumulation". A systematic error in its measurement could lead to violation of conservation-of-packets (instability) or forever remaining in congestion epoch (beat-down/under-utilization). The errors in accumulation could include effects of in-band measurement, effects of interior queueing delays (present even in underload), and effects of change in system delays (eg: routing, scheduling effects). For this paper, we do not examine issues of control/data packet loss, reverse path congestion, clock skew/drift etc. Techniques like timeouts, windows and GPS references could be used in careful combination with the techniques outlined below. This section outlines the key aspects of the detection methods[9]

Recall that section 3 assumed a *continuous out-of-band* measurement model. In contrast, real measurements happen with *discrete in-band* control packets. To emulate out-of-band control as far as possible, the egress measures output rates over *fixed intervals* T which is used to compute accumulation when the kth control packet arrives (Figure 4(a),(b),(c)). While this is similar to out-of-band *measurement*, it *does not* correspond to out-of-band *feedback*. This is because the accumulation $a_i(k)$ can be calculated only when *both* the estimates $\lambda_i(k)$ and $\mu_i(k)$ are available, i.e. at the later of the two events (k^{th}

[9] The pseudo-code was left out for space reasons.

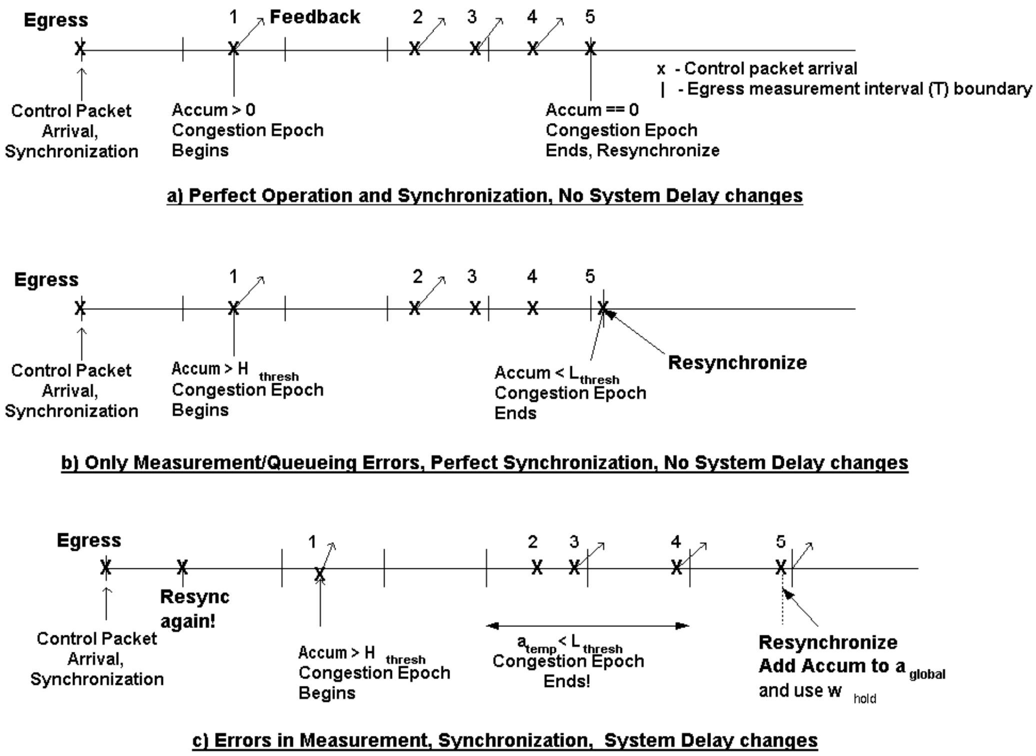


Figure 4: Measurement of Accumulation At the Egress

control pkt receipt, k^{th} egress interval expiry). While we have experimented with aggressive techniques to compensate for this delay during overload, we do not present them for reasons of simplicity. A perfect measurement with *discrete in-band* control packets is shown in Figure 4(a).

Measurement errors may also occur due to synchronization reasons. With in-band control packets, our default assumption is that the egress node synchronizes its measurement intervals with the very first control packet (eg: control pkt 0 in Figure 4(c)). If the network had a non-zero queue at this time, throughout the epoch a real non-zero queue will be mis-interpreted as a zero or negative accumulation. Therefore, it is necessary to add procedures for the accumulation estimate $a_i(k)$ to eventually resynchronize and correlate to the real base queue. Beyond this, we could have *system delay changes* (due to routing, scheduling etc) which result in errors in $a_i(k)$ (eg: see Figure 4(c)) which need to be filtered out. Any residual positive accumulation at the end of an epoch $a_i(k)$ is captured using a "global accumulation" (a_{global}) estimate. These estimation details are summarized below:

- The accumulation $a_i(k)$ is calculated only when *both* the estimates $\lambda_i(k)$ and $\mu_i(k)$ are available, i.e. at the later of the two events (k^{th} control pkt receipt, k^{th} egress interval expiry). To *partially* filter initial synchronization errors, we use a test phase, where the source holds its rate constant at a supposedly "safe" value. If the accumulation goes negative during this phase, it is reset to zero, and the measurement interval is resynchronized to this new control packet (Figure 4(c)).

The test phase is exited when the accumulation is non-negative. Now, since system queues cannot grow without affecting $a_i(k)$ of participating flows, the epoch is detected when $a_i(k) > H_{thresh}$ (Figure 4(b) and (c)) and a boolean variable *Epoch* is set. The policy $min\{\lambda_i(k), \beta\mu_i(k)\}$ now provisions drain capacity corresponding to the incremental accumulation, and the ingress *latches* to the minimum $\beta\mu_i(k)$ over the course of the epoch which ensures consistent drain. The end-of-epoch is detected using L_{thresh} (Figure 4(b)).

- However, since *incremental accumulation could be caused by system delays as well*, we need to go beyond simple hysteresis thresholds to detect end of the epoch. We first *assume that system delay changes are smaller than T between re-synchronization points*. This could be assured by setting T large enough and periodically measuring the edge-to-edge round-trip time. Now the test for the end of an epoch is by comparing a new *short-term accumulation* estimate, a_{temp} with L_{thresh} when the system is close to an epoch end (Figure 4(c)). Let I denote the count of intervals (incremented at the end of intervals) and C denotes the count of control packets (incremented upon receipt of control packets). During the epoch (*Epoch* $== 1$), the condition $I - C \leq 0$ calculated upon receipt of control packets *in successive intervals* portends an end of epoch. The bound on system-delay changes made earlier ensures that $I - C$ will indeed become non-positive. The measure a_{temp} is reset often to filter system delay effects. This also forces the system to either exit the epoch or add more accumulation. The combination of these techniques assures that the system will come out of the epoch if congestion has indeed ended.

- Given these procedures $a_i(k)$ may be positive when congestion ends (Figure 4(c)). To avoid any systematic bias, we capture $max(a_i(k), 0)$ in a_{global} upon re-synchronization:

$$a_{global} = a_{global} + max\{a_i(k), 0\};$$
$$w_{hold} = min(a_{global}, w_{max});$$
$$a_{global} = a_{global} - w_{hold};$$

  w_{hold} is a *hold-down window* used to provision drain capacity for any *residual positive accumulation* between re-sync points caused by system delays, measurement and/or initialization errors, i.e., compensate for residual conservation-of-packets errors and filter out system-delay changes. The price paid is transient under-utilization due to the hold-down mechanism. In particular, the ingress enforces a rate-difference w_{hold}/T till the next re-synchronization point. Let us introduce a new input rate *control* parameter is $r_i(k)$ and let $\lambda_i(k)$ is the *measured rate at the input*, $\lambda_i(k)T \leq r_i(k)T + \sigma - w_{hold}$. A simple way to enforce this is to reset $r_i(k) = r_i(k) - w_{hold}/T$, where $\frac{w_{hold}}{T} \leq \frac{w_{max}}{T} < \alpha$, the rate-increase parameter. Observe that since $r_i(k)$ is increased based upon $\lambda_i(k-1)$, and the transmission is regulated by a token-bucket, demand burstiness does not affect the network queues significantly. This also addresses the Use-it-or-Lose-it (UILI) problem of taking away the ingress rate allocation if demand does not exist to use it [9].

6.2 Parameter Configuration

Consider the parameters τ (rate increase interval) and T (measurement interval at ingress/egress). The factors affecting these parameters are: maximum edge-to-edge system delay (excluding queuing delays), maximum changes in system delay, minimum transmission rate and maximum clock skews (all factors relevant between re-synchronization points). Other factors include maximum queue desired, canonical H_{thresh} settings, timer-granularity, overheads (computation and bandwidth) of control traffic. A setting of τ larger than maximum system delay and a lazy adjustment for system changes (using RTT estimates between edges) is assumed in this paper. Both T and τ are assumed to be large compared to minimum transmission time and maximum clock skews (a GPS reference or alternate mechanism could be used to detect/bound skews) to minimize measurement error. In general, it may be preferable to set $T < \tau$ to sample the system often, and obtain rate-decrease indications faster than rate-increases. However, in our simulations, we set $T = \tau$ to the maximum edge-to-edge propagation delay.[10].

The rate increase parameter α can be set to σ/τ to emulate TCP-like window increases. With this setting, it can be shown that the single-bottleneck maximum queue is $O(N\sigma)$ where N is the number of flows through the bottleneck. [11] The rate-decrease parameter β affects the queue drain time, the length of the congestion epochs, and steady state utilization. Higher values of β may increase the coupling between bottlenecks and hence increase the queue sizes indirectly. For example in a linear network [27] if the bottlenecks get congested alternately, then the long flow *could* be forever in congestion. So to prevent this we could set β such that the time taken for a bottleneck to drain (drain time) is lesser than the time taken for flows to cause congestion again (rise time). If we assume the single bottleneck expressions for both the rise time[12] ($\frac{(1-\beta)C}{N_b\alpha}$) and drain time ($\frac{N_b\alpha}{(1-\beta)C}$), we have: $\frac{(1-\beta)C}{N_b\alpha} > \frac{N_b\alpha}{(1-\beta)C}$ Therefore, $\beta < 1 - \frac{N_b\alpha}{C}$. If $\frac{N_b\alpha}{C} = 0.1$, β can be set to around 0.9. In our simulations, we set it to 0.95.

7 Performance Characteristics and Overlay Service Capabilities

This section illustrates basic properties of the building block: multi-bottleneck operation, fairness, effects of queue distribution to edges; and gives simple illustrations of service capabilities.[13]. Since the closed-loop building block is positioned *to complement, and not substitute* the state-of-the-art, our simulations will *only show complementary behavior and not comparative behavior* with schemes like Vegas, ECN, FRED (congestion control)[14] and assured services, guaranteed services (bandwidth services). The common parameter settings used are as follows: Packet size = 1000 bytes, $\beta = 0.95$, $\alpha = 1\,pkt$, $H_{thresh} = 2\,pkts$, $L_{thresh} = 1.5\,pkts$, $W_{max} = 0.2\,pkts$, $T = \tau =$ maximum edge-to-edge propagation delay. Simulations are run till steady state is reached (10-100 seconds usually).

[10] Setting τ based upon individual loop RTTs is also admissible (see section 4, last para)

[11] Proof is a simple extension of that given in Section 4.

[12] Can be derived from Equation (12)

[13] A larger simulation suite; *ns* code; and Linux-prototypes will be made publicly available after the review period

[14] This also emphasizes the distinction between "end-to-end" and "edge-to-edge"

7.1 Performance Characteristics of the Building Block

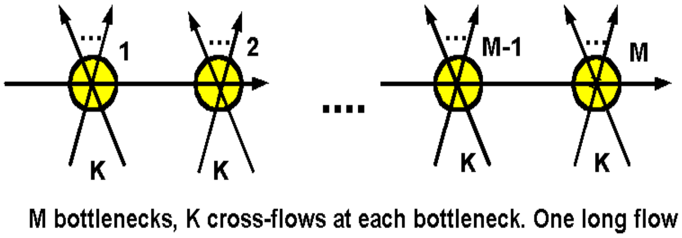
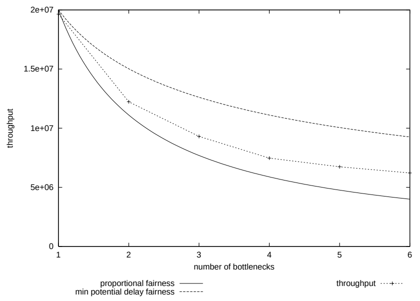
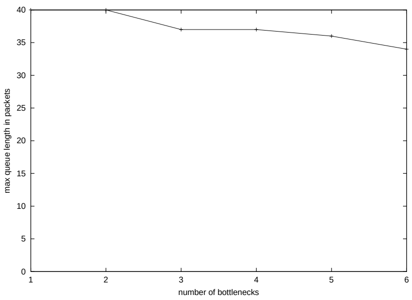


Figure 5: Linear Network Configuration to Illustrate Fairness

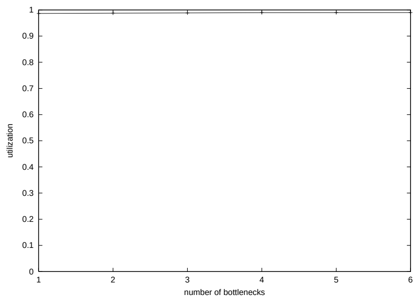
Consider the linear multi-bottleneck network with cross-flows shown in Figure 5, a well-known configuration to illustrate proportional fairness [27]. We vary the number of bottlenecks M from 1 to 6, and the number of cross-flows K is 4. Bottleneck capacities C are 100 Mbps each. Figure 6(a) shows the throughput achieved by the long flow compared against the theoretically computed values for proportional fairness (lowest curve) and min-potential delay fairness (highest curve) [27, 22, 23]. Observe that the throughput achieved (middle curve in Figure 6(a)) lies between the two fairness curves, closer to proportional fairness. The utilization in all cases (Figure 6(c)) is very close to 100% and the maximum of all bottleneck queue lengths (Figure 6(b)) in every case is less than 40 packets. No packets are lost in the network. The subsequent sections describe services using the building block.



Fairness Curves vs Simulated Throughput



Max Queue Lengths

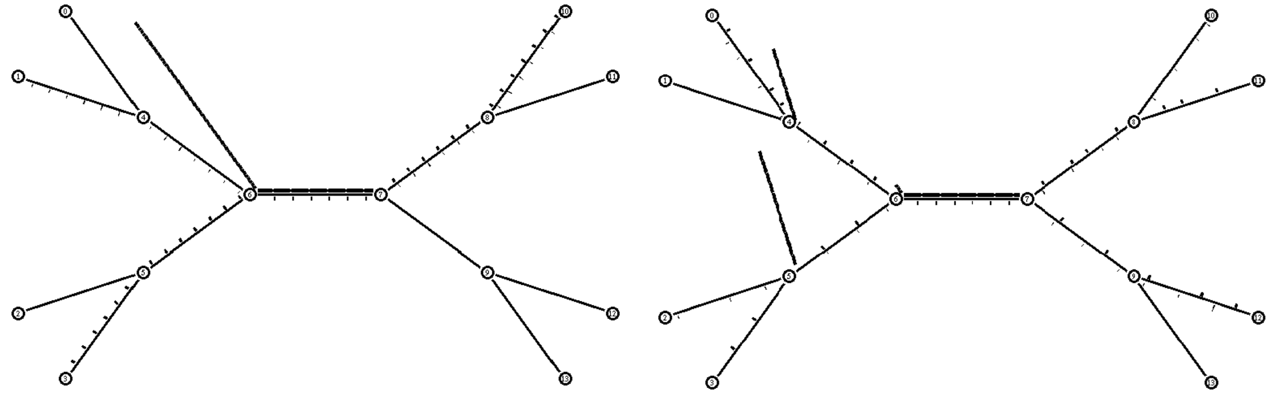


Utilization

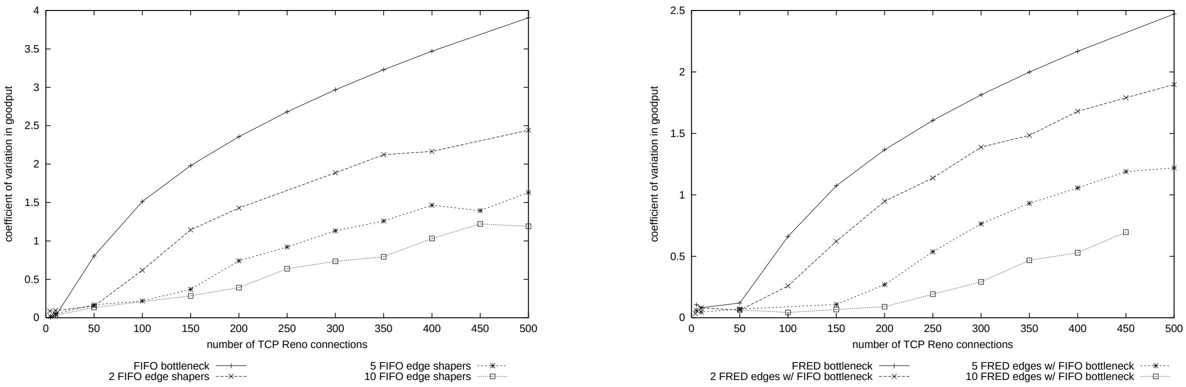
Figure 6: Illustration of Proportional Fairness Characteristics of the Building Block

7.2 Distributed Buffer Management

The simplest service which can be implemented by the overlay framework is *enhanced* best-effort service. The use of edge-to-edge loops simply distributes aggregate backlog among edge-nodes where a range of buffer management options can be used to customize the final loss assignment to users. By utilizing potentially unused edge buffers, it lets the network handle a larger number of flows during overload. The basic behavior is illustrated in Figures 7(a),(b), a *nam* snapshot: N end-to-end TCP microflows (in this case 4 flows) are spread over E edge-to-edge loops (in this case 2 loops). Without the edge-to-edge loop operation, the entire queue builds at node 6 (Figure 7(a)). With the introduction of edge-to-edge loop control, the queue and associated queue management decisions are distributed among E-nodes 4 and 5 (Figure 7(b)), and node 6 operates with a small steady state queue/full utilization. Except for aggregate level isolation and sufficient buffers, no support is expected from node 6.



a) Queue at Bottleneck b) Queue distributed across edges

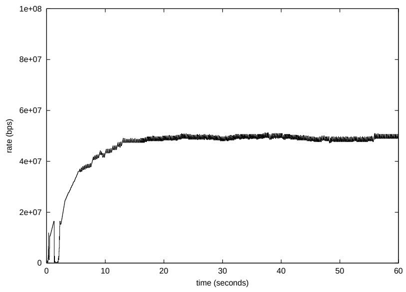


c) FIFO b/neck vs Loops+FIFO Edges d) FRED b/neck vs Loops+FRED Edges

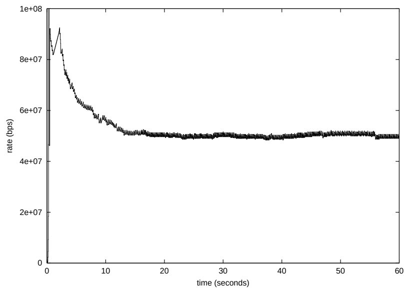
Figure 7: Effect of Queue Distribution on End-to-End TCP Performance

Figures 7(c),(d) extend this study for N ranging from 1 to 500 and $M \in \{0, 2, 5, 10\}$. The bottleneck capacity C is 10 Mbps and all buffers are sized at one bandwidth-delay product (i.e. 50 pkts). Beyond 50 flows, the number of flows is larger than the bandwidth delay product, an extreme case of congestion. The graph plots coefficient of variation ($CoV = \frac{\sigma}{\mu}$)[15] of the *per-TCP goodputs* against the number of TCP flows, and using two queue management algorithms (FIFO and FRED [26]). The CoV is a measure of the *spread* of best-effort service, and the goal is to minimize spread while maximizing the mean per-TCP goodput. The mean per-TCP goodput in all cases was close to 100%, but CoV can vary (in the 500 flow case) from $CoV = 0.7$ to $CoV = 4$! The biggest performance boost is achieved by the simple, transparent distribution of queues away from node 6. Observe that for the 500 flow case with just 2 edge-to-edge loops and FIFO everywhere (i.e. the curve which reaches $CoV = 2.5$, in

[15] μ, σ here denote the mean and standard deviation of per-TCP goodput



(a) TCP Reno (Well-Behaved)



(b) Misbehaving UDP Flood

Figure 8: Edge-to-edge Building Block Isolates Misbehaving Flows at Edges

Figure 7(c)) performs on par with FRED at the interior bottleneck (the curve which reaches $CoV = 2.5$, in Figure 7(d)). The performance of FIFO everywhere improves dramatically with the addition of more edge-to-edge loops ($CoV = 1.2$ with 10 loops+FIFO). Use of FRED at the edge lowers CoV down to 0.7. The intent of this illustration is not to suggest *substitution* of an AQM scheme like FRED with FIFO, but to recognize the additional scalability (factor of 4 improvement in CoV) offered by *complementing* FRED at the edge with edge-to-edge control.

We have experimented with numerous AQM schemes, TCP-aware techniques [16, 26] and TCP-ECN [31] at the edge complemented with edge-to-edge loops and found *consistent order-of-magnitude improvements in CoV, and significant net reduction in overall loss rate and TCP timeouts.* This suggests that TCP *end-to-end loops do not interact negatively with edge-to-edge aggregate control, and the net benefit is indeed very positive.* In general, application-aware or even active/programmable techniques can be used at the edge, possibly in cooperation with the end-system to customize on an application-by-application basis.

7.3 Isolating Misbehaving Flows

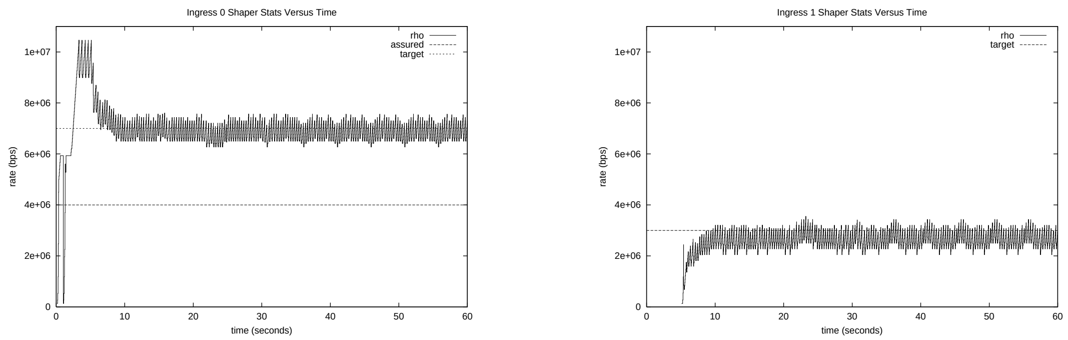
Consider a simple misbehaving flow scenario involving one bottleneck and two end-to-end flows: one UDP and one TCP. The use of overlay control can isolate the UDP flow's excess traffic *at the edge* and offer a fair share to TCP (see Figure 8). In a more general scenario, the edge-to-edge *aggregate* (which may include some misbehaving micro-flows) will be offered its fairshare in competition with other edge-to-edge flows. The *misbehavior* of micro-flows within the aggregate *does not affect other edge-to-edge flows.* Further fine-grained isolation and penalty-box techniques can be performed at the edge [14, 16, 36, 25]. This edge-based isolation capability could be extended to provide model-based isolation support for new experimental end-to-end schemes (eg: Vegas, pricing [22, 8, 28]).

7.4 Assured Service and Quasi-Leased Line Emulation

The assured service [10, 5] can be emulated on our overlay framework by backing off the portion of the ingress departure rate that exceeds the assurance, A_i. In other words, the new rate *decrease* policy is:

$$\lambda_i(k+1) = \min\left(A_i + \beta(\mu_i(k) - A_i),\ \gamma\mu_i(k),\ \lambda_i(k)\right) \quad \text{where } \beta < \gamma < 1$$

In Figure 9 we show results from a simulation using the single bottleneck configuration used in Section 7.3 except the bottleneck is 10Mbps and the TCP connection has 4Mbps of assured bandwidth. Notice that the TCP connection obtains its 4Mbps assurance plus half of the remaining 6Mbps for a total of 7Mbps. The above result assumed no over-subscription of the bottleneck in terms of *assured* rates. With over-subscription, this service simplifies to a *relative priority* service where one class of traffic competes much more aggressively for bandwidth than another. The difference between this service and regular assured service, frame-relay CIR/PIR service and simple priority is that the packet losses are ultimately assigned at the edges (where more intelligence can reside for service customization during contention) and not at the interior bottleneck. As mentioned in section 5, we can also vary the increase/decrease parameters (α, β) to offer *weighted proportional fair services* [23, 22, 12, 30].



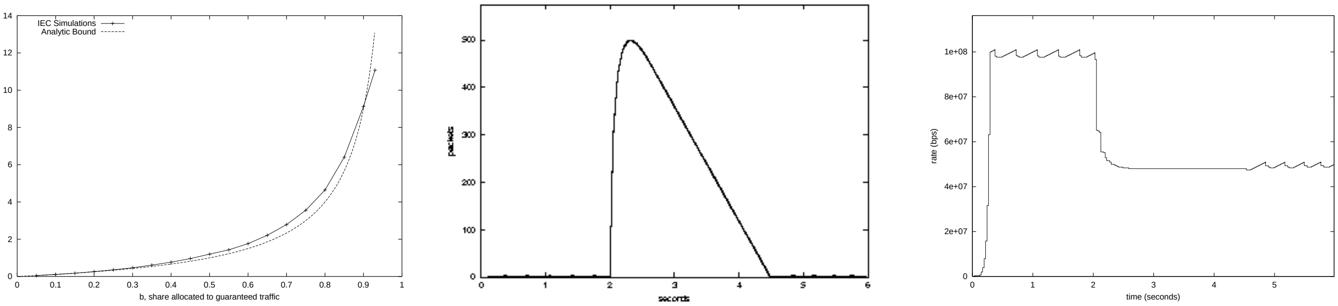
(a) TCP w/ 4Mbps Assured + 3Mbps Best-Effort (b) UDP w/ 3Mbps Best-Effort

Figure 9: Assured Service for TCP versus UDP

With the addition of admission control, edge-to-edge control can provide rate guarantees as well as quick convergence for premium loops that are guaranteed minimum capacity. More specifically, a source can suddenly send at a guaranteed bitrate without a linear increase/slow-start phase, and without backing off once congestion is detected upto its guaranteed rate. This service assures peak bandwidth and near-zero loss, but does not provide jitter or delay assurances. This service is called the *Quasi-leased line (QLL)*. The backoff policy is:

$$\lambda_i(k+1) = max\{A_i,\ A_i + \beta(\mu_i(k) - A_i)\}$$

Observe that unlike the assured service emulation, the QLL service cannot be over-subscribed. If this QLL service is over-subscribed, the above control policy will become unstable. Hence some form of edge-based admission control is required. The service behavior is illustrated again with a simple example of two-loops, one best-effort, and one QLL. At time t = 2 seconds, the QLL flow appears and starts at 50 Mbps. The rate graph (Figure 10(c)) shows the *background best-effort flow* backing off quickly, allowing the QLL to reach its minimum guarantee. The loss-less assurance in the QLL service depends upon the buffer provisioning since a large transient queue develops (Figure 10(b)). A simple *single-bottleneck* analysis[16] yields a queue bound of $\mathsf{qmax} = \frac{b}{1-b}$, which increases dramatically beyond b = 0.5. Figure 10(a) shows the simulated queue bound with the scheme closely matches this simple analysis.



Analytic vs Simulated QLength | Bottleneck queue length | Rate of Best-effort Background Flow

Figure 10: Behavior of a virtual link and QLL for $b = 0.5$

8 Discussion and Limitations of the Overlay Framework

Complex deployment issues of bandwidth services in inter-provider networks were the motivation behind this work. This section examines residual issues and limitations of the framework.

Design Philosophy: The edge-to-edge operation is inspired by the end-to-end principle which limits the coordination required to a pair of edges. While the scheme does not place *implementation or upgrade requirements* at bottlenecks, it does place *configuration requirements* (eg: aggregate isolation, minimum buffers, minimum bandwidth) which can be satisfied by static methods (eg: SNMP), lightweight signaling [6] or policy control [13] mechanisms. The scheme *does not assume VC-based, connection-oriented or symmetric paths* and hence can be overlaid on top of a range of networks (IP, MPLS, ATM, Frame-relay etc).

Loop Setup/Mapping Issues: The *loop-setup* and mapping of end-to-end flows to edge-to-edge flows can be done by initially exchanging the site's set of CIDR address prefixes among other sites. When a packet arrives, a classification into an edge-to-edge aggregate can be done based upon its source-destination prefix-pair, compared to the sites' prefix-list pairs. Loops need to be setup only between pairs of sites belonging to a VPN or extranet. Thus, a site of company A does not setup a loop with a site of company B, avoiding N^2 meshing problems seen in BGP-4 and other routing protocols.

[16] Excluded for space reasons

Incremental deployment can be performed on a customer-by-customer basis. When the sites of a new customer are identified and routes mapped, the *potential* bottlenecks on the path must configured with a separate overall class and buffers (a new class is not required if one already exists). Note that over-engineered clouds are *not considered potential bottlenecks*.

Limited Heterogeneity: In an inter-provider scenario, paths/potential bottlenecks may be mapped by looking at BGP-4 and IGP routing information within each provider's network. For bandwidth assurances, additional configuration information must be exchanged to ensure proper provisioning. Though the information exchange is small, it is not zero, and hence may limit the degree of heteregeneity supported by the framework.

Scalability: The framework *scalability* depends upon the maximum multiplexing of loops N_b at any bottleneck (affects buffer size), the number of loops managed by any edge, number of loop-setups required for all VPNs supported by an ISP, and the number of potential bottlenecks to be reconfigured. A subset of this control-plane complexity could be centralized at a policy server [13], directory or management system.

Rate- vs Window-control: The building block uses rate-based control because it allows a simple thresholding scheme on the "accumulation." However, it can be further enhanced by *window-control and timeouts* to strictly enforce the conservation of packets principle. Control packets in this case will be *bipolar*, i.e., they will dictate both increases and decreases of rate-/window-parameters, and could be used to piggyback clock skew estimation information. One can also imagine hybrid schemes by combining the candidate congestion indicators: packet loss, accumulation ($\sum_j q_{ji}(t)$), network delay ($\sum_j \frac{q_j(t)}{C_j(t)}$) etc. Such hybrid techniques could tolerate a greater spectrum of vagaries, and could reduce buffer reqts for *fully* lossless operation. Our current work is focussed on understanding the inter-relationships between these control parameters and congestion indicators under highly variable network conditions. We are implementing these various alternatives in a Linux-based testbed and will report experimental and comparative results soon.

In summary, the isolation requirement, limited heterogeneity support and limited scaling characteristics of the building block excludes it from general end-to-end use. The technique *does not* solve *public* internetwork congestion control or QoS problems. However, within the scope of private networking on the Internet, the framework can support a useful (though limited) spectrum of services and application-level service customization. Further, it can potentially serve as a platform for pricing schemes (eg: [22]), dynamic point-to-set or programmable services (eg: [10]). These represent a rich set of "*edge-to-end*" problems for future study.

References

[1] Agrawal R., et al, "Performance Bounds for Flow Control Protocols," *ACM/IEEE ToN,* Vol. 7, No. 3, June '99, pp. 310–323.

[2] Awduche D., et al, "A Framework for Internet Traffic Engineering," *Internet Draft, Work-in-progress*, July 2000.

[3] Balakrishnan H., et al, "An Integrated Congestion Management Architecture for Internet Hosts," *SIGCOMM '99*, Aug '99.

[4] Bansal D. and Balakrishnan H., "Binomial Congestion Control Algorithms," *INFOCOM 2001*, Apr '01.

[5] Blake S., et al., "An Architecture for Differentiated Services" *RFC 2475*, Dec '8.

[6] Braden R., et al,"Resource Reservation Protocol (rsvp) - V1 Functional Spec.," *RFC 2205*, Sep '97.

[7] Braden R., et al, "Integrated Services in the Internet Architecture: an Overview," *RFC 1633*, June '94.

[8] Brakmo, L., and Peterson L., "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE JSAC,* Vol 13, No. 8 (Oct '95) pp. 1465-1480.

[9] Charny A. and Ramakrishnan K., "Time-scale Analysis and Scalability Issues for Explicit Rate Allocation in ATM Networks," *IEE/ACM ToN*, Vol. 4, No. 4, Aug '96.

[10] Clark D. and Feng W., Explicit Allocation of Best-Effort Packet Delivery Service, *IEEE/ACM ToN*, Vol. 6, No. 4, Aug '98, pp. 362-373.

[11] Clark D., et al, "NETBLT: a bulk data transfer protocol," *SIGCOMM, '87* pp.353-359.

[12] Crowcroft, J., and Oechslin, P., "Differentiated End-to-end Internet Services using a Weighted Proportional Fair Sharing TCP," *CCR*, Vol. 28, No. 3, July 1998.

[13] Durham D., et al, "The COPS (Common Open Policy Service) Protocol," *RFC 2748*, Jan '00.

[14] Floyd S. and Fall K., "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM ToN*, Aug '99.

[15] Floyd S., et al, "Equation-Based Congestion Control for Unicast Applications," *SIGCOMM '00*, Aug '00.

[16] Floyd S. and Jacobson V., "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM ToN*, Vol. 1, No. 4, Aug '93, pp. 397–413.

[17] Guerin R. and Peris V., "Quality-of-Service in Packet Networks: Basic Mechanisms and Directions." *Computer Networks,* Vol. 31, No. 3, Feb '99, pp. 169-179.

[18] Huston G., "Internet Performance Survival Guide: QoS Strategies for Multiservice Networks," *John Wiley, ISBN 0471-378089*, Jan '00.

[19] Jacobson, V., "Congestion avoidance and control," *SIGCOMM '88*, Aug '88, pp. 314–329.

[20] Jacobson V., et al, "An Expedited Forwarding PHB," *RFC 2598*, June '99.

[21] Jain R., "A delay-based approach for congestion avoidance in Interconnected Heterogeneous Computer Networks," *ACM CCR,* Vol 19, No. 5, Oct '89, pp. 56-71.

[22] Kelly F., et al, "Rate control for communication networks: shadow prices, proportional fairness and stability," *J. of Opn. Research Soc.*, Vol.49, No. 3, Mar '98, pp. 237-252.

[23] Kunniyur S. and Srikant R., "End-to-end congestion control schemes: utility functions, random losses and ECN marks," *INFOCOM '00.*

[24] Kung H.T. and Morris R., "Credit-Based Flow Control for ATM Networks," *IEEE Network Magazine,* Vol. 9, No. 2, March/April '95, pp. 40-48.

[25] Lefelhocz C., et al, "Congestion control for best-effort service: Why we need a new paradigm," *IEEE Network*, Vol. 10, Jan '96, pp. 10-19.

[26] Lin D. and Morris R., "Dynamics of Random Early Detection," *SIGCOMM '97*, Aug '97.

[27] Massoulié L., and Roberts J., "Bandwidth sharing: objectives and algorithms," *INFOCOM '99*.

[28] Mo J., et al, "Analysis and comparison of TCP Reno and Vegas," *INFOCOM'99*, Apr '99.

[29] Mo,J., and Walrand J., "Fair end-end Window based Congestion Control," *IEEE /ACM ToN*, vol.8, '00, pp. 556-567.

[30] Nandagopal T., et al, "Scalable service differentiation using purely end-to-end mechanisms: Features and limitations," *IwQoS '00*.

[31] Ramakrishnan K., Floyd S., "A Proposal to add Explicit Congestion Notification (ECN) to IP," *RFC 2481*, Jan '99.

[32] Ramakrishnan K.K. and Jain R., "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM TOCS*, Vol. 8, No. 2, May '00, pp. 158-181.

[33] Rosen E., et al, "Multiprotocol Label Switching Architecture," *RFC 3031*, Jan '01.

[34] Savage S., et al, "TCP Congestion Control with a Misbehaving Receiver," *ACM CCR*, Vol 29, No. 5, Oct '99.

[35] Schulzrinne H., et al, "RTP: A Transport Protocol for Real-Time Applications," *RFC 1889*, 1997.

[36] Stoica I., et al, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks," *SIGCOMM '98*, Aug '98.