Edge-To-Edge Traffic Control for the Internet

David Harrison, Shivkumar Kalyanaraman

Abstract

We propose a new overlay architecture and algorithms to augment best-effort congestion control in the Internet called *edge-to-edge traffic control*. The basic architecture works at the network (IP) layer and involves pushing congestion back from the interior of a network, distributing it across edge nodes where the smaller congestion problems can be handled with more stateful and sophisticated methods. The *edge-to-edge virtual link* thus created can be used as the basis of several applications. These applications include controlling TCP and non-TCP flows, improving buffer management scalability, and isolating bandwidth-based denial-of-service attacks. The methods are flexible, combinable with other protocols (like MPLS and diff-serv), and can be incrementally deployed.

1 Introduction

We propose an architecture where control loops operate between domain edges to regulate the aggregate traffic between each edge-pair. Each control loop creates what we call a *virtual link*. The operation of the architecture involves the exchange of control packets on a per-edge-to-edge virtual link basis. To construct virtual links we develop a new set of congestion control techniques which break up congestion at interior nodes and distribute the smaller congestion problems across the edge nodes. We thus leverage the buffers at the edge nodes during congestion to increase the effective bandwidth-delay product of the network. Further, these smaller congestion problems can be handled at the edge with existing buffer management, rate control, or scheduling algorithms. By combining virtual links with other building blocks, we can isolate bandwidth-based denial of service attacks, offer simple differentiated services, and introduce new dynamic contracting and congestion-sensitive pricing schemes. Most importantly, we can build all of these applications with minimal complexity placed in the interior (less than RED [8]).

In [9] we report initial results on a scheme that requires no interior support. We will publish more complete results of this advancement in our next paper. Also, we have simulated mechanisms for weighting virtual links such that each obtains a share proportional to the number of flows passing through the virtual link or to satisfy aggregate bandwidth assurances. We will publish these results soon. In this paper we focus on the use of virtual links to augment best-effort congestion control for TCP and UDP.

This paper is organized as follows. Section 2 presents the new congestion control concepts behind our scheme. Related work is discussed in section 2.3. Section 3 introduces the edge-to-edge overlay architecture and presents the details of our scheme. Section 4 samples simulation results illustrating the performance of the core scheme. Section 5 describes the use of EEC for distributed buffer management. Section 6 discusses how edge-to-edge control relates to other congestion solutions. We present detailed control flow diagrams and an analysis of queueing behavior in the appendices.

2 Core Congestion Control Concepts

Our scheme is based upon the simple observation that *at all times*, the sum of the output rates of flows passing through a particular bottleneck ($\sum \nu_i$) is less than or equal to the bottleneck capacity μ (see Figure 1). Most important, this condition holds during periods of congestion called "congestion epochs".

2.1 Congestion Epoch and Increase/Decrease Policy

Our definition of congestion epoch is any period when the instantaneous queue length exceeds a queue bound which is larger than the maximum steady state queue fluctuations. The steady-state queue fluctuation is limited by requiring each sender generate traffic constrained to a Linear Bounded Arrival Process (LBAP)[7]. Chosen this way, the congestion epoch is the period of full utilization incurred when the mean aggregate load (λ) exceeds mean capacity (μ). Observe that our notion of a congestion epoch does not involve packet loss in its definition and is a basis for "early" detection.

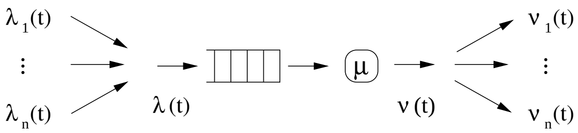
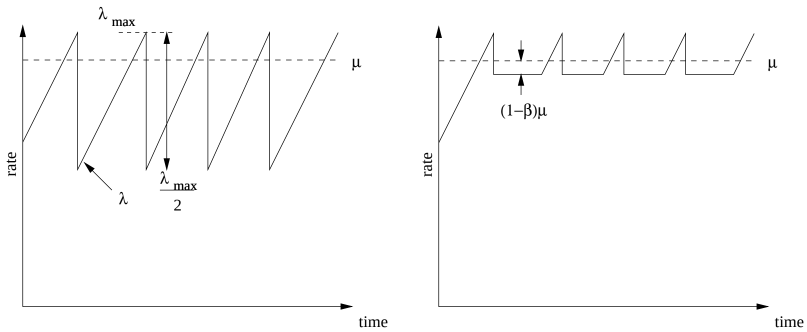


Figure 1: Traffic Control Model

The output rates of flows (ν_i) can be measured at the receiver and fed back to the sender. During congestion epochs each sender imposes a rate limit r_i such that $r_i \leftarrow \beta \nu_i$ where $\beta < 1$. If each sender *consistently* constrains its input rate (λ_i) such that $\lambda_i \leq r_i$ during the congestion epoch, the epoch will eventually terminate. This is intuitively seen in an idealized single bottleneck, zero-time delay system because the condition $\sum \beta \nu_i < \sum \nu_i \leq \mu$ causes queues to drain. In the absence of congestion, we employ additive increase to probe for the bottleneck capacity limits.

Observe that our increase-decrease policy is not the same as the well known additive-increase multiplicative-decrease (AIMD) policy, because our decrease policy is based upon the output rate (ν_i) and not the input rate (λ_i). We therefore refer to our policy as AIMD-ER (Additive Increase and Multiplicative Decrease using Egress Rate). We retain the multiplicative aspect in the decrease policy because it promotes fairness as shown by Chiu and Jain [4]. Furthermore, a schematic of our scheme's steady state input rate dynamics (Figure 2) looks different from an equivalent one for TCP or other AIMD schemes. Specifically, the schematic for AIMD schemes resembles a saw-tooth whereas our scheme holds a constant rate while the queue drains.



AIMD AIMD-ER

Figure 2: Steady-State Dynamics

2.2 Congestion Epoch Detection

The remaining part of the basic approach is a method to detect the congestion epochs in the system. Our scheme assumes that *the bottleneck assists in the determination of the start and duration of a congestion epoch.* Specifically, the bottleneck promiscuously marks a bit in departing packets whenever the instantaneous queue length exceeds a carefully designed threshold.

2.3 Related Congestion Control Work

The core ideas which form the scheme basis are not completely new. But our definition of "congestion epochs" and response strategy is unique to the best of our knowledge. The observation that output rates could be used as a basis of congestion response was first used in the design of NETBLT[6]. However, NETBLT negotiates rates only once per block transfer and congestion is detected based on rate difference. Unfortunately rate difference is prone to instability when rate differences are small.

The concept of the congestion epoch and early congestion detection is different from that of TCP since our concept does not require packet loss for congestion detection. The concept is similar to work done in DECbit [20] and ATM [21][13], but the combination of mechanisms and policies used to realize it differ as follows:

- First, although we use observation intervals at the egress and bit-marking to declare congestion epochs, we do not attach any significance to the *number* of packets seen with bits marked (unlike the DECbit scheme [20]).

- Second, we use *bit-indications in the forward direction and explicit rate feedback in the reverse direction* which is a hybrid of techniques used in bit-based schemes (TCP-ECN [19], DECbit [20]) and explicit rate schemes (ATM ABR) [21][13]. Note that the latter schemes use either bit-based or explicit rate feedback in *both* directions.

- Third, though our decrease policy is similar to AIMD, *our decrease is applied to the measured output rate ν_i*. Thus the multiplicative factor used in our scheme can be safely chosen to be larger than 0.5 (which is the factor chosen in TCP).

Our virtual link concept parallels TCP trunks [3], which uses TCP-style AIMD for congestion control. However TCP trunking still relies on loss to detect congestion and thus our schemes can respond earlier to congestion.

The following sections describe the edge-to-edge overlay architecture which we propose to build out of our basic mechanisms and the details of the two schemes developed within this architecture.

3 Edge-to-edge Overlay Congestion Control Architecture

For this paper, we assume the network is partitioned into *traffic control domains*. A traffic control domain is a set of networks with consistent traffic control policies. Specifically, we assume that edge-to-edge controlled traffic is isolated from other traffic which is not edge-to-edge controlled. Our architecture has three primary components: the *ingress edge*, the *interior router* and the *egress edge*. Nodes within a traffic control domain that are connected to nodes outside the domain and implement edge-to-edge control are known as edge routers. Routers otherwise within a domain are called interior routers (see Figure 3).

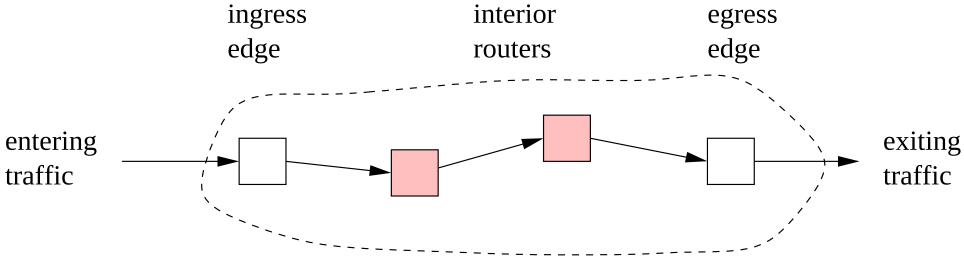


Figure 3: Traffic Control Domains

Using the scheme outlined in section 2, the ingress acts as the sender and the egress the receiver. Interior routers promiscuously mark when congestion occurs. If the bottleneck is an edge then more sophisticated congestion mechanisms might be employed but we ignore this possibility here. Because this scheme relies on explicit congestion indication, we call this the Explicit Edge Control (EEC) scheme.

Since the ingress acts as a sender, it must constrain all traffic entering each virtual link to a LBAP descriptor, $\int_t^{t+\tau} \lambda_i(T)dT \leq r_i\tau + \sigma$ where r_i is the ith virtual link's rate limit and σ is the maximum burst size. The ingress can implement this constraint by subjecting all traffic entering a virtual link to a (σ, ρ)-shaper [7] (a.k.a., a leaky bucket shaper).

In a single bottleneck configuration as shown in Figure 4 with n virtual links, the maximum queue length is $n\sigma$ as long as λ does not exceed μ. Thus we provision some $N >\approx n$ and set the bottleneck mark threshold to $N\sigma$. A rough estimate of N suffices and could be based on periodic edge-initiated tracerouting of the paths taken by virtual links through the domain. More direct means are available where permanent virtual circuits or label-switched paths are available in the network.

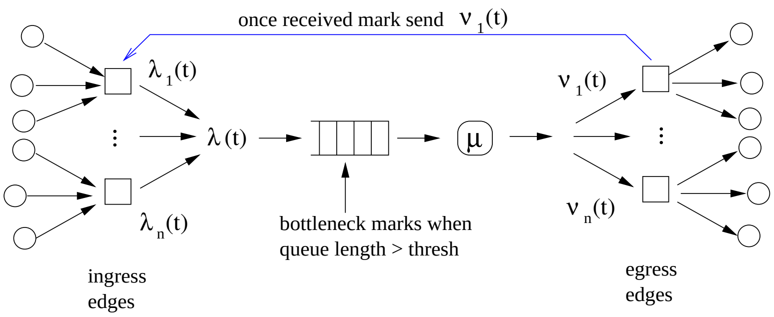


Figure 4: Ingress aggregates end-to-end flows into virtual links. When queue length exceeds threshold, all packets are marked. Upon receiving mark, egress generates negative feedback.

3.1 Scheme Details

Implementation of our control scheme has some additional notable details:

- All edges agree on a measurement interval T.
- During additive increase, the rate increments by $\frac{\sigma}{\mathsf{T}}$ at the beginning of each interval.
- Intervals are resynchronized with the arrival of the first marked packet signalling the beginning of a congestion epoch.
- As long as marked packets arrive in an interval, the egress sends one control packet per interval to the ingress. Each control packet bears the smoothed egress rate ν_i.
- The virtual link resumes additive increase when the ingress receives no control packet from the egress for two intervals.
- The virtual link uses periodic keep-alives in the absence of feedback to handle lossy return channels.
- In addition to AIMD-ER, each virtual link i implements a rate-based slow-start up to a threshold, $rthresh_i$. $rthresh_i$ is analogous to TCP's SSTHRESH.

The dynamics of EEC are illustrated in Figure 5.

The measurement interval T used by *all* edge systems (both ingress and egress) is set to the domain-wide maximum edge-to-edge round-trip propagation and transmission delays, max_ertt, plus the time to mark all virtual links passing through the bottleneck when congestion occurs. The time to mark all virtual links can be roughly estimated as $\frac{N_{max}\sigma}{\mu_{min}}$ where μ_{min} is the smallest capacity within the domain and N_{max} is a reasonable upper bound on the number of virtual links passing through the bottleneck. Because all virtual links use the

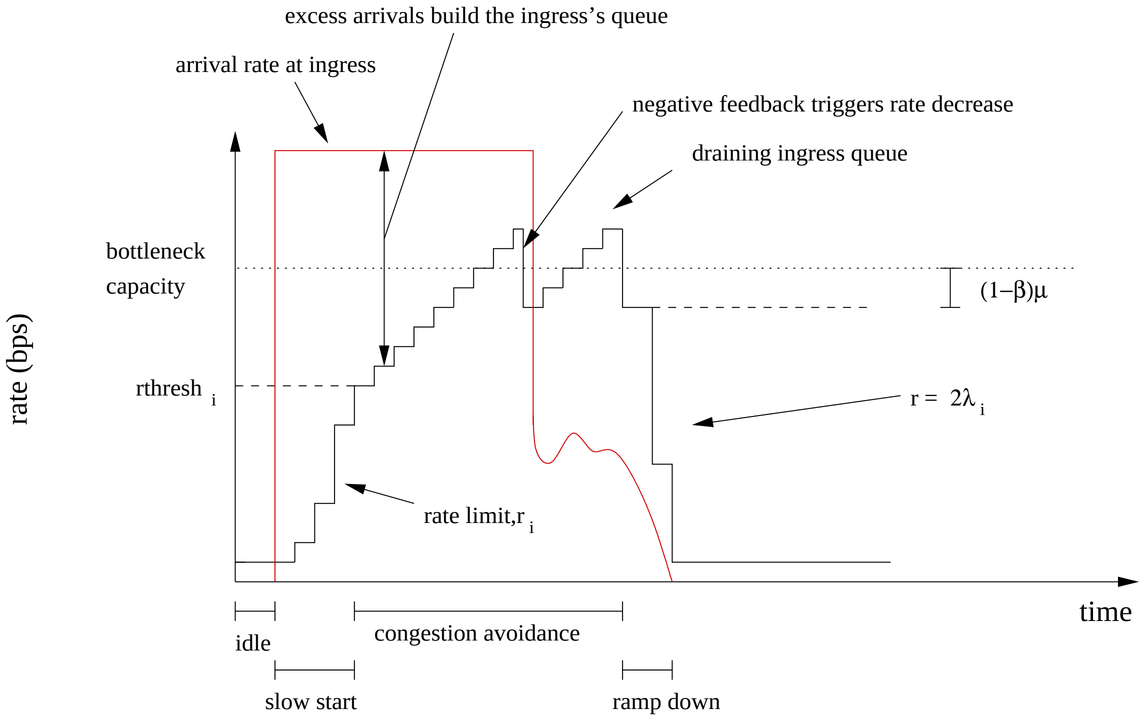


Figure 5: Detailed Dynamics of Edge-to-edge Rate Regulation

same interval, they increase with roughly the same acceleration and will all backoff within T of being marked. We show in Appendix A that our scheme achieves a *steady state queue length bound* of approximately $4N\sigma$ for a single bottleneck with fixed capacity and persistent demand. The bound is not a function of RTT partly due to the fact that the rate limit increases by at most σ/T in every interval T, thus "acceleration" varies with the inverse of fixed delays.

To improve fairness in the system, the scheme optionally delays backoff through a method known as "delayed feedback response." Specifically, the feedback received by the ingress node is enforced after a delay of $max\_ertt - ertt_i$, where $ertt_i$ is the edge-to-edge round trip of the i-th virtual link. This step attempts to roughly equalize the time-delay inherent in all the feedback loops of the traffic control domain.

If the domain has large heterogeneity in its delays such that setting T to the maximum edge-to-edge RTT results in undesirably long feedback loops, each virtual link can set its T_i to some smaller value so long as each interval is greater than or equal to the virtual link's edge-to-edge round-trip time. As a result, shorter control loops tend to grab slightly more share than longer control loops. Keep in mind that a single domain may be an ISP network with a much shorter edge-to-edge round trip time than the end-to-end round trip. In such cases, decreasing the edge-to-edge control loop merely degrades fairness without improving performance.

4 Performance of Edge-to-edge Control

We simulated our schemes using `ns-2.1b5`. With a few exceptions noted where appropriate, we use the parameters shown in Figure 6.

bottleneck buffer size	1.5 $\mathsf{T}\mu/\sigma$ packets[a]
data packet size	1000 bytes
max burst, σ	8000 bits
interval[b], T	rtp + $N\sigma/\mu$
backoff, β	0.98
mark threshold	N packets

[a]For comparison against TCP without edge control, we use 1.5 bandwidth-delay (i.e., end-to-end delay) product worth of buffer in sections 4.2 and 4.4.

[b]rtp=max edge-to-edge round-trip propagation delay

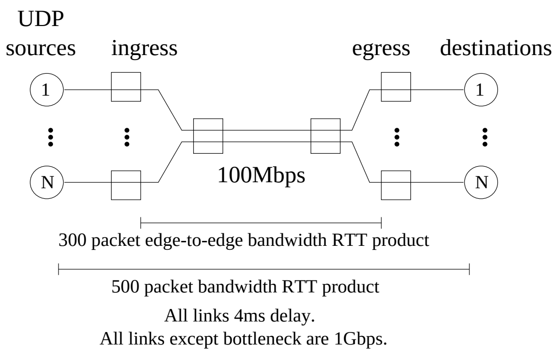


Figure 6: Simulation parameters and Single Bottleneck Topology

4.1 Single bottleneck

These simulations demonstrate EEC controlling persistent UDP flows[1]. Figure 7 shows that EEC stays within the analytically-derived steady-state queue length bound $4N\sigma$ (derived in Appendix A), has high utilization, low mean bottleneck queue lengths, experiences zero drops in the interior, and remains fair to within a standard deviation that is 5% of the mean (i.e., coefficient of variation < 0.05). Further, these properties hold true across a wide range (2-100) in the number of virtual links passing through the bottleneck.

Virtual[a] Links, N	Utilization	Queue Length[b] (avg/max pkts)	Drops[c] (pkts)	C.O.V. Throughput[d]
2	98.8%	0.821/6	0	0.0236
5	98.6%	3.43/15	0	0.0083
10	98.3%	8.58/31	0	0.0373
20	97.5%	20.1/62	0	0.022
100	96.4%	104.9/279	0	0.0448

[a] Statistics collected starting at 10 seconds into a 30 second simulation.
[b] Bottleneck queue length. We ignore ingress queue length for now.
[c] Only drops in interior during steady state. UDP incurs substantial loss at the edges.
[d] Coefficient of Variation of Throughput$=\frac{\text{standard deviation}}{\text{mean}}$

Figure 7: Single 100Mbps Bottleneck with UDP Sources

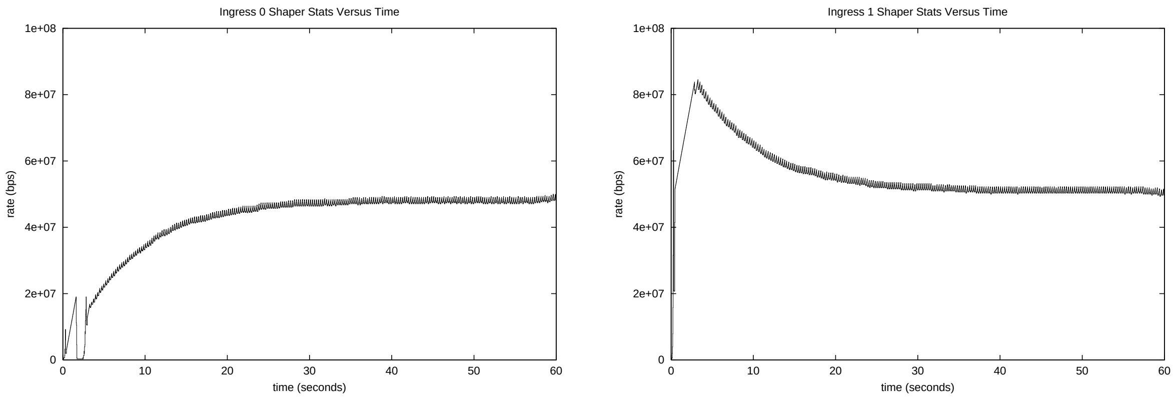
4.2 Misbehaving Flows

Consider a denial of service attack where a UDP flow floods the bottleneck with packets forcing any TCP connections passing through the bottleneck to a trickle. This is demon-

[1] Sending at four times the fair share.

strated in Figure 9 where an attacking *UDP flow constrains a TCP connection to 1200bps of the 100Mbps bottleneck*. EEC can isolate bandwidth-based denial of service attacks at the edge. As shown in figures 8 and 9, EEC constrains the UDP flow close to its fair share (i.e., 49.9 Mbps).

Figure 8 also shows that initially TCP lags. This occurs because TCP's end-to-end feedback loop is larger than the edge-to-edge control loop. In other words, the edge-to-edge virtual link adapts more quickly to demand than TCP adapts to available capacity. As a result, UDP initially provides all the demand and is given capacity that *TCP cannot immediately use*.



TCP Reno connection | UDP flow sending at 200Mbps

Figure 8: TCP versus UDP. Both regulated by Explicit Edge Control (EEC)

4.3 Heterogeneous Delay

We show two scenarios with EEC pitted against a FIFO and a FRED bottleneck. In the first EEC scenario, the heterogeneity is inside the traffic control domain. In the second EEC scenario, the heterogeneity is outside the traffic control domain[2]. In both scenarios, EEC is

[2]labelled in Figure 10 as "Inverse scenario w/ EEC + FIFO."

Bottle-neck[a]	Edge Scheme	Utilization	Queue Length[b] (avg/max pkts)	Drops (pkts)	Throughput (TCP/UDP in Mbps)
FIFO	EEC	98.7%	0.818/6	1/372901/**0**[c]	**47.9/50.8**
FRED[d]	none	100%	42.5/175	351224[e]	40.5/59.5
FIFO	none	100%	373.5/375	250004[f]	**1200bps/100.0**

[a]Statistics collected starting at 40 seconds in a 60 second simulation.

[b]Bottleneck queue length. We ignore ingress queue length for now.

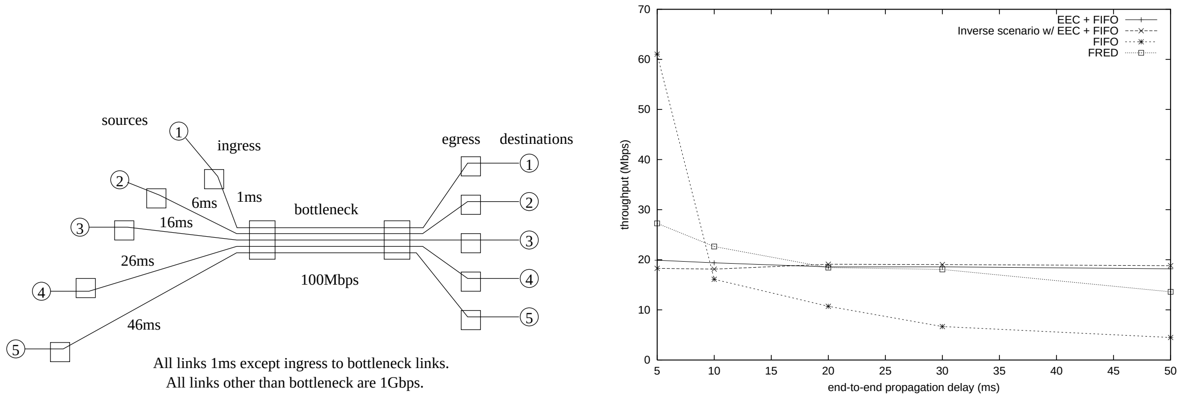
[c]Drops in ingress serving TCP, ingress serving UDP, and interior respectively.

[d]In all simulations we used the default FRED parameter set given in [15].

[e]Drops at the bottleneck. Mostly UDP drops else TCP would receive less share.

[f]Drops at the bottleneck. Mostly UDP drops because TCP is nearly starved.

Figure 9: TCP versus UDP statistics

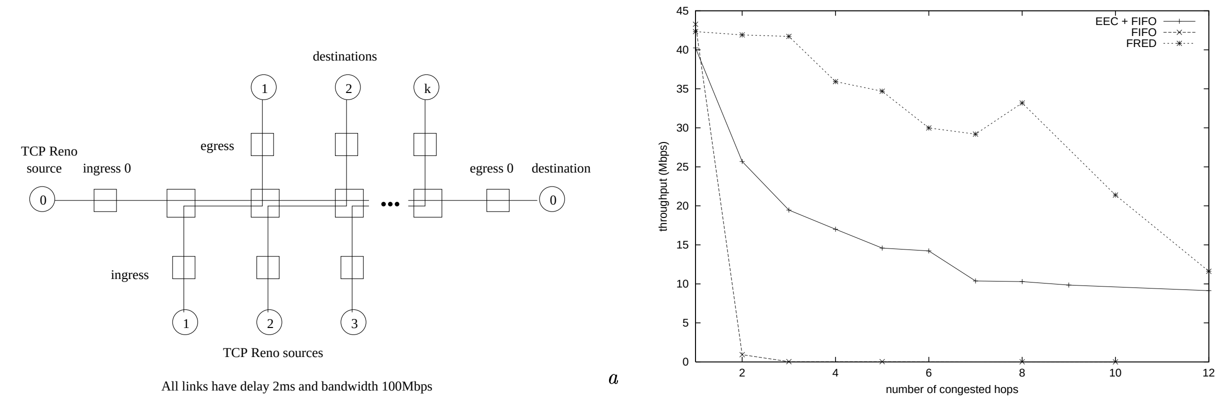


[a]

[a]150 second simulation. Gathered statistics over last 100 seconds

Figure 10: EEC distributes capacity evenly between TCP Reno connections whether delay is inside or outside the domain.

fairer than FRED and substantially fairer than FIFO.



[a]

[a]60 second simulation. Gathered statistics over last 30 seconds.

Figure 11: Reno over EEC experiences less beat-down than Reno over FIFO

4.4 Multiple Bottlenecks

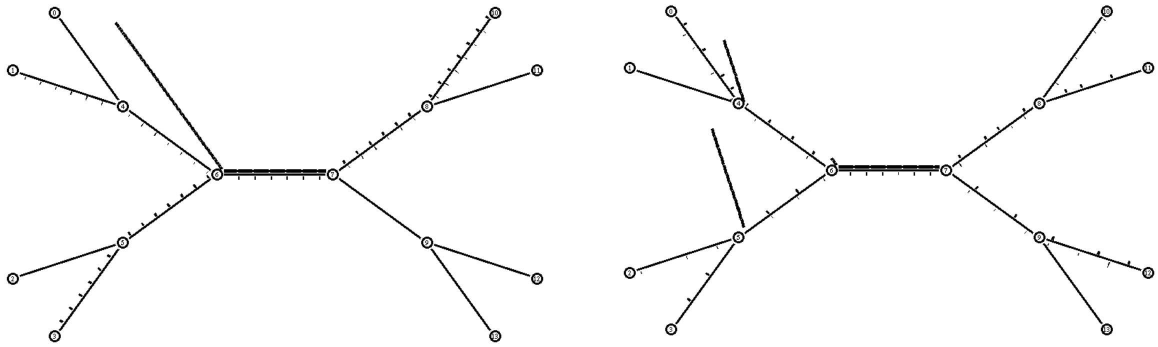
When subjected to a series of bottlenecks, EEC virtual links receive less share than cross-traffic passing through a single bottleneck, a.k.a., beat-down. However, it experiences substantially less beat-down than non-edge-controlled TCP Reno connections subjected to a series of FIFO bottlenecks. The graph in Figure 11 shows that a single TCP Reno connection passing through multiple FIFO's *rapidly approaches zero throughput once the topology contains three bottlenecks.* EEC performs somewhat worse than FRED, but note that FRED requires substantially more complex buffer management at the bottleneck.

5 Distributed Buffer Management

As already stated, edge-to-edge control can be used to distribute backlog across the edges (see Figure 12). This increases the effective number of buffers allowing more TCP connections to obtain large enough windows to survive loss without timing out. This reduces TCP's bias against tiny flows and thus improves fairness [15]. As demonstrated in Figure 13(a), using

EEC to distribute the backlog dramatically reduces the coefficient of variation in goodput[3] when many TCP connections compete for the bottleneck. As expected this improvement increases as we distribute the congestion across more edges.

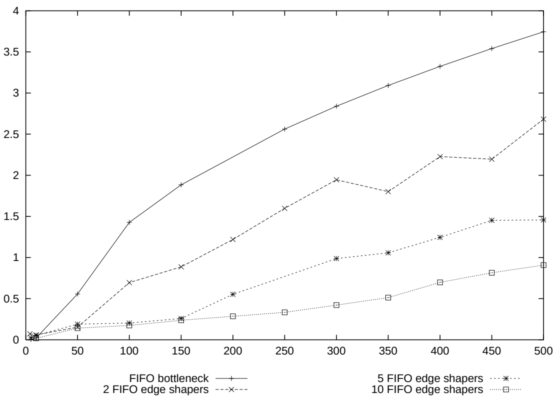
FRED handles large numbers of flows much more effectively than FIFO by preferentially dropping packets from larger flows. However, as demonstrated by the top curve in Figure 13(b), even FRED grows unfair when the number of flows exceeds the number of packets in the bandwidth-delay product plus the number of buffers. With EEC, we can increase the effective number of buffers available to FRED by replacing FRED in the core with a promiscuous marking queue and implementing FRED at the head of each virtual link (i.e., at each ingress edge). Replacing FRED in the core with a promiscuous marker also demonstrates our ability to reduce the complexity imposed on the bottleneck. As expected, the coefficient of variation in goodput improves in some cases by as much as an *order of magnitude*.



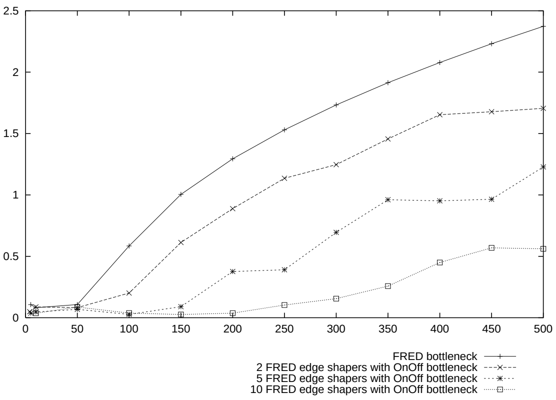
(a) queue at bottleneck (b) queue distributed across edges

Figure 12: Backlog moved from Bottleneck to Edge

[3]Goodput is defined here as the number of transmitted payload bits excluding retransmissions per unit time.



(a) FIFO bottleneck versus EEC+FIFO edges [a] [b]



(b) FRED bottleneck versus EEC+FRED edges [a] [b]

[a] 60 second simulation. Statistics gathered from beginning to end.

[b] 10Mbps bottleneck. Vary number of edges in Figure 12. One bandwidth-delay product worth of buffer.

Figure 13: Distributed buffer management improves fairness under many TCP connections

6 Discussion

Several strategies used to combat congestion include:

1. provisioning more bandwidth at congestion points
2. deploying buffer management (RED[8], FRED[15])
3. web caching
4. upgrading TCP implementations (TCP/SACK [11])
5. bandwidth management (TCP Rate Control [14])
6. MPLS [2]
7. deploying differentiated services

To a certain extent all of these complement each other. Edge control allows complex buffer management algorithms to be deployed along the edges rather than in the interior.

Web caching operates at a higher layer and would thus benefit from improvements in lower layer transport. Upgrading TCP end-systems represents a longer-term solution because of the shere number of end-systems. Policy-based bandwidth management mechanisms [14] typically do not scale to the interior. Edge control allows these mechanisms to manage bandwidth for bottlenecks occurring anywhere within the control loop. MPLS allows ISPs to distribute traffic across multiple paths. Edge control can better control the traffic within a label-switched path. DiffServ [1] provides policy-based service differentiation at every hop. Because of explicit support in the interior, DiffServ provides rich Service Level Agreements (SLAs). For example, if the Expedited Forwarding Per-Hop Behavior is implemented as specified in [18, 12] then the associated premium service will achieve tighter jitter and delay bounds than achievable with edge control but at the cost of over-provisioning and little ability to adapt to available capacity. Because edge control incurrs zero loss over an interior with properly provisioned buffers, edge control might interoperate with the DiffServ Assured Forwarding Per-Hop Behavior [10, 5] by implementing all drop preference at the edge.

The first mentioned and simplest approach, provisioning more bandwidth, works well in ISP backbone networks because of the relatively small number of links to upgrade and the rapid decrease in dollar per bps provided by optical networking. However, anecdotal evidence suggests that congestion points exist in ISP networks exactly where bandwidth cannot be provisioned quickly enough:

- access links
- aggregation points in local or central office Points of Presence (PoPs)
- international links
- tail circuits (to remote locations)

- peering points

We hope to alleviate such congestion by deploying control loops around these trouble spots. Assuming edge control is deployed along with a scheduled upgrade, the most immediate benefit comes in the control and isolation of misbehaving flows. Distribution of congestion becomes important once the demand again creates a bottleneck. Edge control fairly distributes capacity under heavy loads allowing a graceful degradation in performance, extending the life of deployed equipment, and buying time in which to manage the next upgrade.

Controlling congestion at peering points either requires coordination across ISPs in order to overlay control loops through the peering point or requires placing an edge at the peering point. Both approaches could distribute the congestion across access or aggreation points, but the latter approach would likely not work at peering points between more than two ISPs unless the congestion typically occurs at a link immediately connected to the edge router.

7 A Partial Deployment Strategy

EEC edges can be deployed one customer at a time if the interior is a Class-of-Service (CoS) network[1] so that a separate class can be set aside to isolate edge-controlled traffic from other best-effort traffic. This is necessary because loss-based (late) congestion control techniques incur large backlogs. EEC backs off when there is significant backlog allowing other traffic to capture capacity from the edge-controlled class. This same incompatibility occurs when TCP-Reno competes with TCP-Vegas [17].

In addition EEC requires that the bottleneck implement a marker. Some implementations of RED[8] can be configured to act as a promiscuous marker, by setting $w_q = 1$, $min_{th} =$

max_{th}, max_p to 1.0 and asking the router to mark packets instead of dropping them. But this method cannot be applied where such support is unavailable. We are working to eliminate the need for a marker in the interior.

8 Future Work

This paper merely presents an improved best-effort service. We intend to broaden the service menu enabled by edge-to-edge control into an *edge-based services platform.* In line with this objective, we intend to investigate a wider scope of mechanisms employed at the edge to improve end-to-end best-effort service (e.g.,[14, 16]), enable flow-level service differentiation, and extend policy-based bandwidth-management to include remote bottlenecks (e.g., [14]). Our current focus is to implement EEC in a local and then ISP testbed.

References

[1] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. An architecture for differentiated services. IETF RFC 2474, December 1998.

[2] Callon, R., and et al. A framework for multiprotocol label switching. Internet Draft, draft-ietf-mpls-framework-02.txt, November 1997.

[3] Chapman, A., and Kung, H. Traffic management for aggregate IP streams. Tech. rep., Nortel Networks, 1999.

[4] Chiu, D., and Jain, R. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks 17*, 1 (June 1989), 1–14.

[5] Clark, D., and Fang, W. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking 6*, 4 (August 1998), 362–373.

[6] Clark, D. D., Lambert, M. L., and Zhang, L. NETBLT: A high throughput transport protocol. In *SIGCOMM Symposium on Communications Architectures and Protocols* (August 1987), pp. 353–359.

[7] Cruz, R. L. *A Calculus for Network Delay and a Note on Topologies of Interconnection Networks.* PhD thesis, University of Illinois, July 1987.

[8] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (August 1993), 397–413.

[9] Harrison, D., and Kalyanaraman, S. Edge-to-edge traffic control for the internet. Tech. rep., Rensselaer Polytechnic Institute, http://networks.ecse.rpi.edu/˜harrisod/, April 2000.

[10] Heinanen, J., Finland, T., Baker, F., Wiess, W., and Wroclawski, J. Assured forwarding phb group. Internet RFC 2597, June 1999. Defines per-hop behaviors building blocks for assured services.

[11] Jacobson, V., and Braden, R. Tcp extensions for long-delay paths. IETF RFC 1072, October 1988. Discusses TCP SACK.

[12] Jacobson, V., Nichols, K., and Poduri, K. The "virtual wire" behavior aggregate. Internet Draft, ftp://ftp.ee.lbl.gov/papers/vw_ba.pdf, March 2000.

[13] Kalyanaraman, S., Jain, R., Fahmy, S., Goyal, R., and Vandalore, B. The ERICA switch algorithm for ABR traffic management in ATM networks. *IEEE/ACM Transactions on Networking 8*, 1 (February 2000), 87–98.

[14] Karandikar, S., Kalyanaraman, S., Bagal, P., and Packer, B. TCP rate control. *Computer Communication Review 30*, 1 (2000).

[15] Lin, D., and Morris, R. Dynamics of Random Early Detection. *SIGCOMM'97* (August 1997).

[16] Lo Monaco, G., Feroz, A., and Kalyanaraman, S. Tcp friendly marking for scalable and "better" best-effort services on the internet. Submitted to ICNP 2000, November 2000.

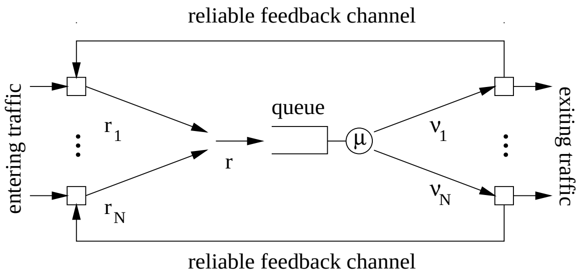
[17] Mo, J., La, R. J., Anantharam, V., and Walrand, J. Analysis and comparison of TCP Reno and Vegas. In *Proceedings of IEEE INFOCOM '99* (March 1999).

[18] Nichols, K., Jacobson, V., and Zhang, L. A two-bit differentiated services architecture for the internet. Internet Draft, ftp://ftp.ee.lbl.gov/papers/dsarch.pdf, November 1997.

[19] Ramakrishnan, K. K., and Floyd, S. A proposal to add Explicit Congestion Notification (ECN) to IPv6 and to TCP. Internet draft, November 1997.

[20] Ramakrishnan, K. K., and Jain, R. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *Proceedings of ACM SIGCOMM '88* (August 1988), pp. 303–313.

[21] Siu, K., and Tzeng, T. Intelligent congestion control for ABR service in ATM networks. *ACM Computer Communication Review 24*, 5 (October 1995), 81–106.



$r(t)$	aggregate rate limit
$q(t)$	queue length
$I[t, t+\tau]$	traffic entering in $[t, t+\tau]$
T	interval length
N	number of flows
σ	max burst size
a	acceleration $= \frac{\sigma}{\mathsf{T}^2}$
t_μ	time when $r(t) = \mu$
$N\sigma$	mark threshold

Figure 14: Topology, Constants, and Variables used in Analysis

A Steady-State Queue Length Bound

We use a simplified fluid model to derive a heuristic bound on the bottleneck queue length for the Explicit Edge Control(EEC) scheme. In this model, we use the topology in Figure 14. For simplicity, throughout this analysis we use the term *flow* to denote a virtual link. The model has the following properties and assumptions:

- when $q(T) > N\sigma$, the queue marks all departing fluid thus marked fluid does not go through the queue
- prior to backoff, the throttle increases rate limit by at most constant acceleration a
- $\mathsf{T} >$ maximum edge-to-edge round trip propagation and transmission delays
- time is in units of intervals T
- we do NOT assume homogeneous delay

Each flow is shaped by a leaky bucket shaper. With N flows, the amount of traffic I arriving in $[t, t+\tau]$ becomes:

$$I[t, t+\tau] \leq \int_{t}^{t+\tau} r(T)dT + N\sigma \tag{1}$$

Prior to a congestion epoch, when the aggregate input rate first reaches $r = \mu$, the persistent queue length is zero. The rate limit starting from t_μ up to the time when the queue begins draining obeys:

$$r(t) \leq Na(t - t_\mu) + \mu \tag{2}$$

The bottleneck queue after an interval $[t, t+\tau]$ in which the queue does not reach zero is:

$$q(t+\tau) = q(t) + I[t, t+\tau] - \mu\tau \tag{3}$$

Let the queue begin marking at $t = 0$. Thus at $t = 0$ we know that the queue length is $N\sigma$. The interval T bounds the time before the bottleneck begins to drain. Thus in time units of T, we know that the maximum queue length is less than or equal to the worst-case queue length at $t = 1$.

$$q(1) = N\sigma + I[0,1] - \mu \leq N\sigma + (\frac{Na}{2} + r(0) + N\sigma) - \mu \tag{4}$$

All terms appearing on the right-hand-side of inequality (4) are constant except $r(0)$, thus max $q(1)$ occurs with max $r(0)$. Max $r(0)$ occurs when r accelerates as fast as possible in $[t_\mu, 0]$ with no burst causing the queue length to prematurely reach the threshold. Thus,

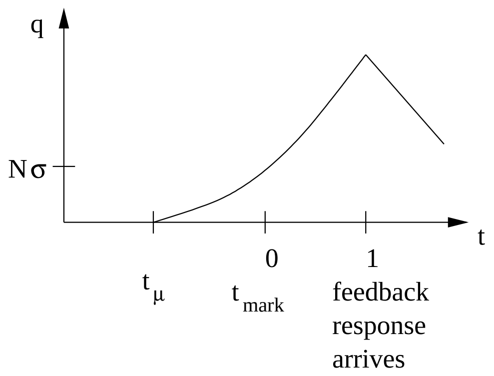


Figure 15: Queue length versus time

$$\int_{t_\mu}^{0} Na(T - t_\mu)dT = N\sigma, \quad t_\mu = -\sqrt{\frac{2\sigma}{a}} \tag{5}$$

$$r(0) = N\sqrt{2\sigma a} + \mu \tag{6}$$

Combining equation 6 and inequality 4 yields:

$$q(1) \leq \frac{Na}{2} + N\sqrt{2a\sigma} + 2N\sigma = \left(\frac{5}{2} + \sqrt{2}\right)N\sigma \tag{7}$$

$$\text{max q} \leq q(1) < 4N\sigma \tag{8}$$

B Explicit Edge Control Algorithm

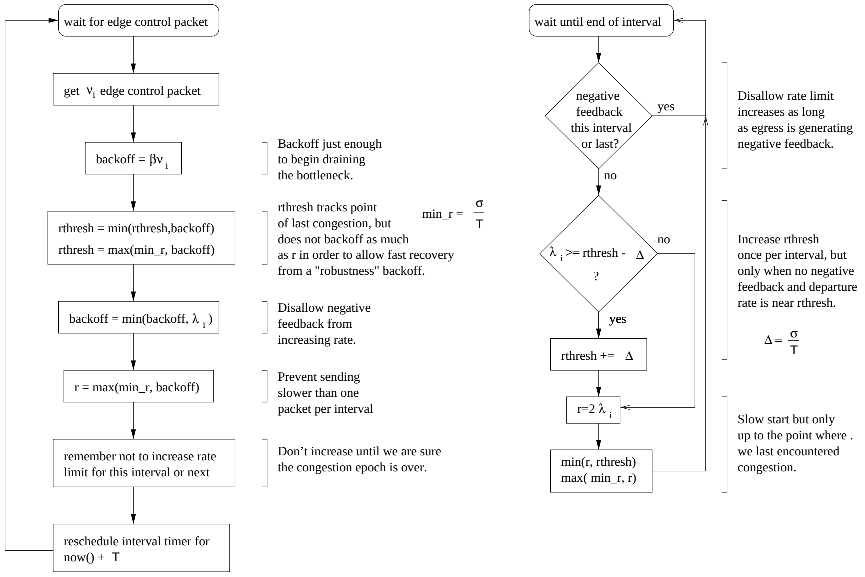


Figure 16: Ingress Control Flow Diagram for EEC

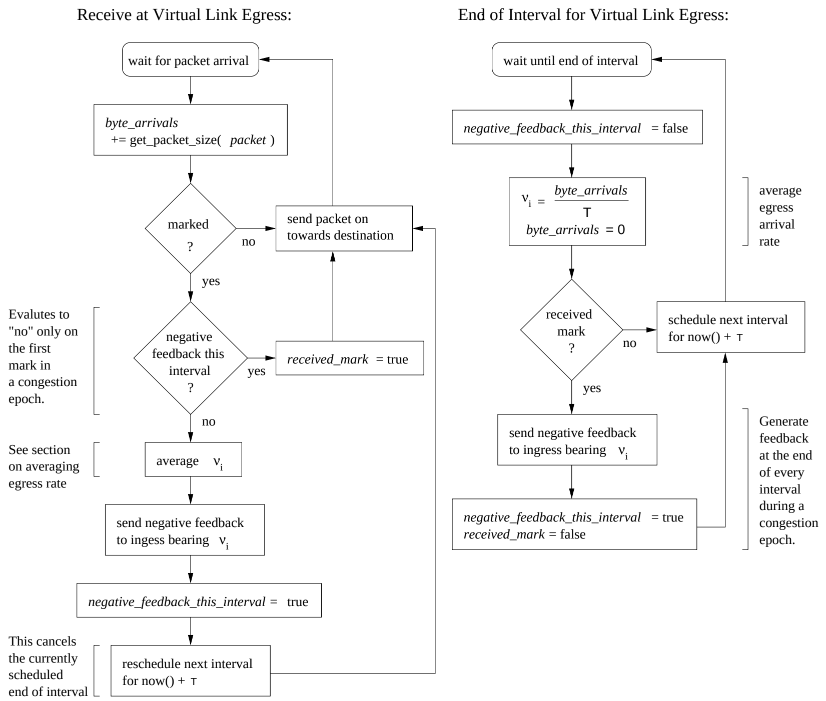


Figure 17: Egress Control Flow Diagram for EEC