Edge-To-Edge Traffic Control for the Internet

David Harrison[1], Shivkumar Kalyanaraman[2]

[1]Computer Science Department

[2]Electrical, Computer, and Systems Engineering Department

Rensselaer Polytechnic Institute

harrisod@cs.rpi.edu, shivkuma@ecse.rpi.edu

January 2000

Updated July 9, 2000

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 flexible, sophisticated and cheaper 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, developing simple differentiated services, and isolating bandwidth-based denial-of-service attacks. The methods are flexible, combinable with other protocols (like MPLS and diff-serv), require no standardization and can be incrementally deployed.

1 Introduction

We propose an architecture where control loops operate between overlay 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. This improves scalability and reduces the cost of these 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 *without upgrading intermediate routers or end-systems*. We illustrate some of these applications in this paper.

This paper is organized as follows. Section 2 presents the new congestion control concepts behind our schemes. Related work is discussed in section 2.3. Section 3 introduces the edge-to-edge overlay architecture and presents the details of our schemes. Section 4 samples simulation results illustrating the performance of the core schemes. Section 5 describes sample applications of the methodology. We present detailed control flow diagrams and an analysis of queueing behavior in the appendices.

2 Core Congestion Control Concepts

Our schemes are 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. 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.

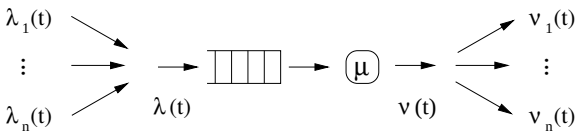
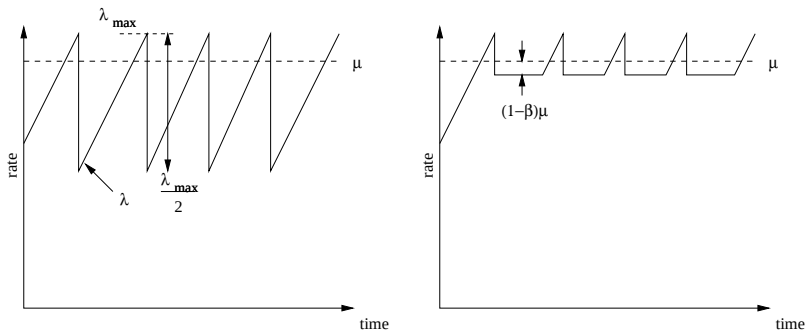


Figure 1: Traffic Control Model

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. We analytically derive a

steady-state queue length bound in appendix C.1.



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. We develop two schemes for this purpose. Our first and simpler scheme assumes that *the interior routers assist in the determination of the start and duration of a congestion epoch.* In our second scheme, *edges detect congestion epochs without the involvement of interior routers.* Specifically, in the first scheme, the interior router promiscuously marks a bit in packets whenever the instantaneous queue length exceeds a carefully designed threshold.

In the second scheme, the edges rely on the observation that each flow's contribution to the queue length (or *accumulation*), q_i, is equal to the integral $\int(\lambda_i(T)-\nu_i(T))\,dT$. If this accumulation is larger than a predefined threshold, the flow assumes the beginning of a congestion epoch. The end of the congestion epoch is detected when a one-way delay sample comes close to the minimum one-way delay.

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. NETBLT[5] is the only

other scheme to the best of our knowledge that uses a policy similar to AIMD-ER, i.e., renegotiating input rate based upon egress measurements. However, NETBLT negotiates rates only once per block transfer and congestion is detected based on rate difference which is prone to instability when rate differences are small. However, the method was not fully developed since the focus of NETBLT at the time was to develop a reliable protocol bypassing the problems of RTT estimation.

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 [17] and TCP Vegas [2], but the combination of mechanisms and policies used to realize it are different:

- First, although we use observation intervals at the egress and bit-marking in one of our schemes to declare congestion epochs, we do not attach any significance to the *number* of packets seen with bits marked (unlike the DECbit scheme [17]).
- 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 [16], DECbit [17]) and explicit rate schemes (ATM ABR) [19][10]. 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).

The use of an individual flow's accumulation to detect congestion is similar to credit-based schemes [12] and the conservation-of-packets principle used in TCP [8]. The novelty of our approach is to apply the conservation-of-packets principle in a rate-based context rather than in a window-based context.

Using delay estimates in relation to the minimum delay to flag the end of a congestion epoch is similar to some of the approaches considered in delay-based schemes introduced by Jain [9] and

the TCP-Vegas work [2]. We observe that one-way delay is difficult to use to detect the *beginning* of a congestion epoch. This is because we know of no reference to determine that changes in delay are sufficient to reliably indicate a congestion epoch. However, the one-way delay estimates can be used to flag the *end of a congestion epoch* because of the existence of such a reference (i.e. minimum one-way delay). Therefore, for the problem of detecting the beginning of congestion epochs, we use the per-flow accumulation concept which *does* have a reliable reference, i.e. the maximum allowed per-flow accumulation.

Our virtual link concept parallels TCP trunks [3], which uses TCP-style AIMD for congestion control. Our use of newer control policies allows earlier congestion response and hence a wider application range.

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

With the edge-to-edge overlay congestion control architecture, control loops (i.e., virtual links) span between edge devices placed at any location within the Internet possibly spanning multiple ISPs provided each ISP agrees to maintain classwise isolation of edge controlled traffic. We expect that most edges for edge control will reside at the customer premise access point behind or integrated with the customer's firewall and virtual links will span between a set of a priori known sites. In this configuration edge-control provides improved service for private networks. The central difference between our architecture and the architecture used by Frame Relay or ATM is that we can operate at L3 thus allowing a virtual link to span multiple service providers using differing data link layers. Further we can do this without upgrades to any ISP network.

3.1 Edge-to-edge Control With Interior Support

In this scheme, the ingress node regulates each edge-to-edge virtual link to a rate limit of r_i. The actual input rate (i.e., departure rate from the ingress, and denoted λ_i) may be smaller than r_i. We also assume that the ingress node uses an observation interval T for each edge-to-edge virtual link originating at this ingress.

A congestion epoch begins when an interior router promiscuously marks a congestion bit on all packets once the instantaneous queue exceeds a carefully designed queue threshold parameter. Since interior routers participate explicitly, we call this the *Explicit Edge Control (EEC)* scheme. The egress node declares the beginning of a new congestion epoch upon seeing the first packet with the congestion bit set. A new control packet is created and the declaration of congestion along with the measured output rate at the egress is fed back to the ingress. The interval used by the egress node to measure and average the output rate is resynchronized with the beginning of this new congestion epoch.

The congestion epoch continues in every successive observation interval where at least one packet from the edge-to-edge virtual link is seen with the congestion bit set. At the end of such intervals, the egress sends a control packet with the latest value of the exponentially averaged output rate. The default response of the ingress edge upon receipt of control packets is to reduce the virtual link's rate limit (r_i) to the smoothed output rate scaled down by a multiplicative factor ($\nu_i \times \beta$; $0 < \beta < 1$). The congestion epoch ends in the first interval when no packets from the virtual link are marked with the congestion bit. The egress merely stops sending control packets and the ingress assumes the end of a congestion epoch when two intervals pass without seeing a control packet.

3.1.1 Scheme Details: Rate Adjustment Policies

The ingress node uses a leaky bucket rate shaper[6], whose rate limit (r) can be varied dynamically based upon feedback. The amount of traffic I entering the network over any time interval $[t, t+\tau]$

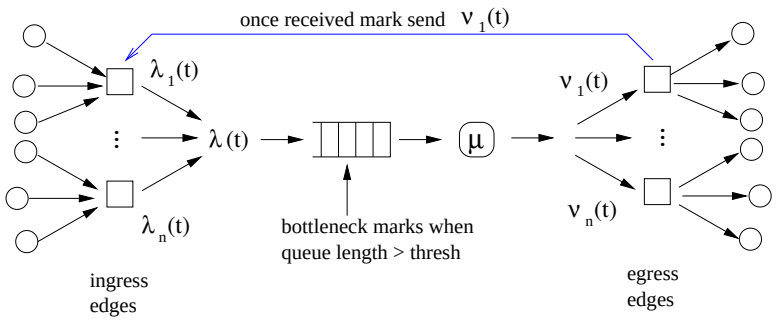


Figure 3: Ingress aggregates end-to-end flows into virtual links. When queue length exceeds threshold, all packets are marked. Upon receiving mark, egress generates negative feedback.

after shaping is:

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

In inequality 1, r is the *dynamic rate limit* and σ is the *maximum burst size* admitted into the network. Assuming that all virtual links are rate-regulated, the queue threshold parameter is set no smaller than $N\sigma$ where N is the number of virtual links (not end-to-end flows) passing through the bottleneck. A rough estimate of N (which suffices for this scheme) can be based upon the number of routing entries, and/or the knowledge of the number of edges whose traffic passes through the node. The idea is to allow at most σ burstiness per active virtual link before signalling congestion.

The initialization of edge-to-edge virtual links occurs in a manner similar to TCP slow start, and we call it "rate-based slow start." As long as there are sufficient packets to send, the rate limit doubles each interval. When a rate-decrease occurs (in a congestion epoch), a rate threshold, $rthresh_i$, is set to the new value of the rate limit after the decrease. The function of this variable is similar to the SSTHRESH variable used in TCP. While the rate-limit r_i tracks the actual departure rate λ_i, $rthresh_i$ serves as an upper bound for slow start. Specifically, the rate limit r_i is allowed to increase multiplicatively till it reaches $rthresh_i$ or receives a congestion notification. Once the

departure rate λ_i (and the rate limit r_i) are close to $rthresh_i$, the latter is allowed to increase linearly (by σ/T) once per measurement interval T. The dynamics of these variables is illustrated in Figure 4.

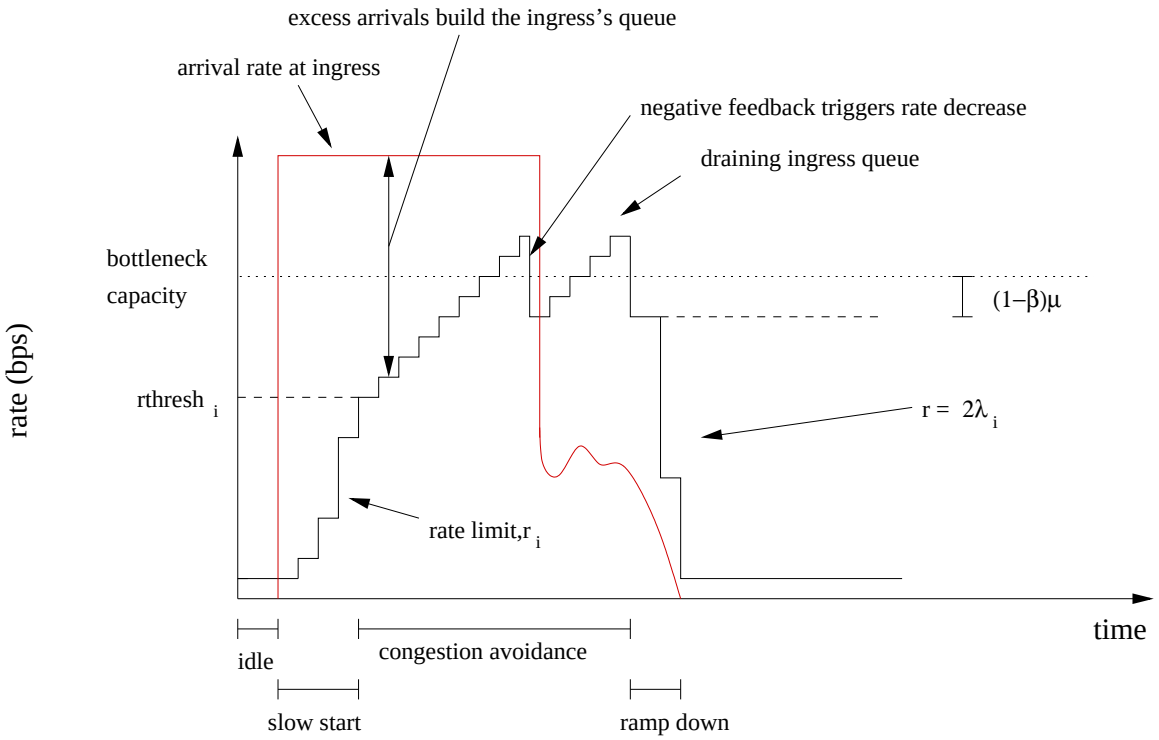


Figure 4: Detailed Dynamics of Edge-to-edge Rate Regulation

The rate-decrease during a congestion epoch is based upon the measured and smoothed egress rate ν_i. The response to congestion is to limit the departure rates (λ_i) to values smaller than ν_i consistently during the congestion epoch. A simple algorithm for this is to limit λ_i by the rate limit parameter $r_i = \beta \times \nu_i, 0 < \beta < 1$ upon receipt of congestion feedback. The rate change (increase or decrease) is not performed more than once per measurement interval T. Moreover when there is a sudden large difference between the load λ_i and the egress rate ν_i, our algorithm adds an additional compensation to drain out the possible queue built up in the interior. A detailed flow chart is presented in Appendix B.

3.1.2 Scheme Details: Robustness Issues

The measurement interval T used by *all* edge systems (both ingress and egress) is provisioned no smaller than the overlay-wide maximum edge-to-edge round-trip fixed delays, max_ertt, plus the approximate 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 overlay and N_{max} is a reasonable upper bound on the number of virtual links passing through the bottleneck. Because all virtual links use the same interval, they increase with roughly the same acceleration and will all backoff within T of being marked. We show in Appendix C.1 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 in the overlay.

If the overlay has large heterogenity in its delays such that setting T to the maximum edge-to-edge RTT results in undesireably long feedback loops, T can be set to some smaller value so long as each feedback loop larger than the overlay-wide T sets its measurement interval equal to its edge-to-edge round-trip time. We can even eliminate the T parameter altogether by having all virtual links use their respective edge-to-edge round trip times. We have shown that this degrades fairness by favoring virtual links with shorter edge-to-edge round trip times (see section 4.3).

3.2 Edge-to-edge Control Without Interior Support

We eliminate the explicit feedback from the interior and call the result *Implicit Edge Control (IEC)*. IEC infers congestion state by estimating the contribution of each virtual link to the queue length q_i by integrating the difference in ingress and egress rates. When the estimate exceeds a threshold, IEC declares congestion. IEC ends the congestion epoch when the delay on a control packet drops to within ϵ of the minimum measured delay. In all other ways, IEC and Explicit Edge Control (EEC) are identical.

3.2.1 Detecting the Beginning of a Congestion Epoch without Interior Support

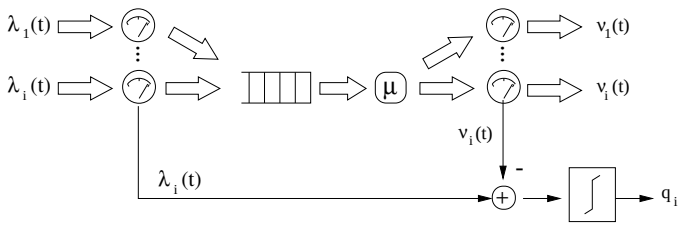


Figure 5: Measuring Virtual Link's Contribution to Queue Length

To detect the beginning of a congestion epoch, each virtual link signals congestion when its contribution ("accumulation"), q_i, to the queue length exceeds σ. When all N virtual links contribute an accumulation of σ, the total accumulation is $N\sigma$ which is the congestion epoch detection criterion used in the EEC scheme. The ith virtual link's accumulation q_i can be calculated using the following observation. Assume a sufficiently large period τ. If the average input rate during a period τ is $\overline{\lambda}_i$, and the average output rate is $\overline{\nu}_i$, the accumulation caused by this flow during the period τ is $(\overline{\lambda}_i - \overline{\nu}_i) \times \tau$. The accumulation measured during this period can be added to a running estimate of accumulation q_i which can then be compared against a maximum accumulation reference parameter. More accurately stated:

$$q_i(t) \quad = \quad q_i(t-\tau) + \int_{t-\tau}^{t} \lambda_i(T)dT - \int_{u-\tau}^{u} \nu_i(T)\ dT \tag{2}$$

$$= \quad q_i(t-\tau) + (\overline{\lambda}_i[t-\tau, t] - \overline{\nu}_i[u-\tau, u])\tau \tag{3}$$

$$u \quad = \quad t + \text{fixed delays} \tag{4}$$

The average interval for $\overline{\nu}_i$ is delayed by the propagation, transmission, processing, and any other fixed delays so that any fluid entering the virtual link by time t can leave by time u unless it is backlogged. As a result the computation of q_i only considers queued packets and not packets in flight.

The ingress node sends two control packets in each interval T (but no faster than the real data rate). τ is the inter-departure time of control packets *at the sender*. In each control packet, the ingress inserts a timestamp and the measured average input rate ($\overline{\lambda}_i$). The average output rate $\overline{\nu}_i$ is measured over the time interval between *arrivals* of consecutive control packets at the egress. The egress node now has all the three quantities required to do the computation: $(\overline{\lambda}_i - \overline{\nu}_i) \times \tau$ and add it to a running estimate of accumulation. The running estimate of accumulation is also reset at the end of a congestion epoch to avoid propagation of measurement errors. We discuss our approximation of q_i in more detail in section B.4.

One way of implementing the control packet flow required for this mechanism without adding extra traffic is for the ingress to piggy-back rate and timestamp information in a shim header on two data packets in each interval T. Interior IP routers ignore the shim headers, while the egress strips them out.

3.2.2 Detecting the End of a Congestion Epoch without Interior Support

The detection of the *end* of a congestion epoch, or in general an uncongested network is based upon samples of one-way delay. As each control packet arrives at the egress, the egress updates the minimum one-way delay seen so far. Every time a one-way delay sample is within ϵ of the minimum one-way delay, the egress declares that the network is uncongested and stops sending negative feedback. Note that the minimum one-way delay captures the fixed components of delay such as transmission, propagation and processing (not queueing delays). The delay over and above

this minimum one-way delay is a rough measure of queueing delays. Since low delay indicates lack of congestion, the scheme does not attempt to detect the beginning of a congestion epoch until a control packet has a delay greater than ϵ above the min delay.

One last point, measuring minimum one-way delay does not require that edges synchronize clock offsets. This is because IEC is interested only in the *delay relative to the minimum delay.* Even negative mininimum delays are allowed. This does not mean that measuring minimum delay is not without its caveats. TCP-Vegas[2] also measures minimum delay. Hence IEC and TCP-Vegas share some problems. These problems and solutions are discussed in [14]. We discuss these issues in more detail in Appendix B.5.

3.2.3 Detecting packet loss

When a bottleneck has underprovisioned buffers or rapidly changes capacity, loss can occur. To handle these conditions, IEC stamps in each control packet the number of packets sent since the previous control packet. The egress signals a packet loss whenever there is a mismatch between the stamped packet count and the number of arrivals at the egress. The IEC virtual link then backs off to $\mu_i/2$.

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.

4.1 Single bottleneck

These simulations demonstrate the EEC and IEC schemes controlling persistent UDP flows[1]. Figure 7 shows that both EEC and IEC stay within the analytically-derived steady-state queue length bound $4N\sigma$ (derived in Appendix C.1), have similar mean bottleneck queue lengths, have high

[1] Sending at four times the fair share.

bottleneck buffer size	1 bandwidth-RTT[a]
ingress buffer size	1 bandwidth-RTT
data packet size	1000 bytes
max burst, σ	8000 bits
interval[b], T	rtp + $N\sigma/\mu$
backoff, β	0.98
mark threshold[c]	N packets

[a] Exception in section 4.1 where we use 1.5 $\mathsf{T}\mu/\sigma$ packets to accomodate many virtual links.

[b] rtp=maximum round-trip propagation delay

[c] applies only to EEC

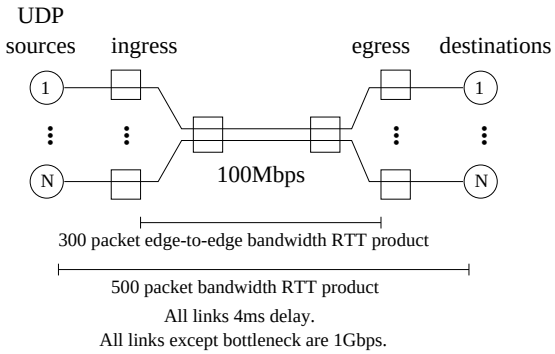


Figure 6: Simulation parameters and Single Bottleneck Topology

utilization, experience zero drops in the interior in the steady state, and remain 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.

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 demonstrated in Figure 9 where an attacking *UDP flow constrains a TCP connection to 1200bps of the 100Mbps bottleneck.* IEC can isolate bandwidth-based denial of service attacks at the edge. As shown in figures 8 and 9, IEC 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

Virtual[a] Links, N	Edge Scheme	Utilization	Queue Length[b] (avg/max pkts)	Drops[c] (pkts)	C.O.V. Throughput[d]
2	EEC	98.8%	0.821/6	0	0.0236
2	IEC	99.0%	0.881/8	0	0.00445
10	EEC	98.3%	8.58/31	0	0.0373
10	IEC	99.2%	9.30/34	0	0.0141
100	EEC	96.4%	104.9/279	0	0.0448
100	IEC	99.2%	120.7/265	0	0.0270

[a]Statistics collected starting at 10 seconds into a 30 second simulation.

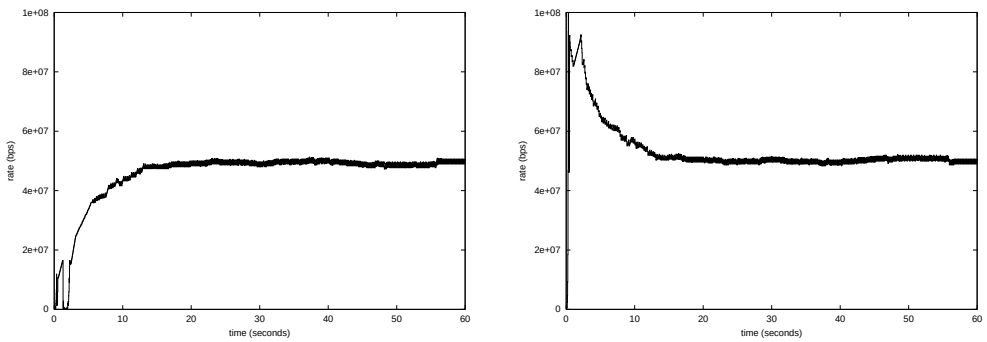
[b]Bottleneck queue length. We ignore ingress queue length for now.

[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

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 Implicit Edge Control (IEC)

4.3 Heterogeneous Delay

We show three scenarios with IEC pitted against a FIFO and a FRED bottleneck. In the first IEC scenario, the heterogeneity is inside the overlay. In the second IEC scenario, the heterogeneity is

Bottle-[a] neck	Edge Scheme	Utilization	Queue Length[b] (avg/max pkts)	Drops (pkts)	Throughput (TCP/UDP in Mbps)
FIFO	IEC	99.0%	0.886/7	2/750322/**0**[c]	**49.1/49.9**
FRED[d]	none	100%	42.5/175	702448[e]	40.5/59.5
FIFO	none	100%	373.5/375	500008[f]	**1200bps/100.0**

[a]Statistics collected starting at 20 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 [13].
[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

outside the traffic control overlay[2]. In the first two scenarios the interval T used by each virtual link is the maximum edge-to-edge RTT. In the third scenario, we once again place the heterogeneity within the overlay, but each virtual link sets its interval equal to its edge-to-edge RTT[3]. Thus T is no longer an input parameter. In all three IEC scenarios, IEC is fairer than FRED and substantially fairer than FIFO. However, there is some degradation in fairness when IEC sets its interval equal to its RTT. As with TCP, IEC without the interval parameter tends to favor virtual links with shorter edge-to-edge round trip times. As an aside, the third scenario (i.e., without a T input parameter) experiences a maximum steady-state queue length of 15 packets[4] and thus does not violate the $4N\sigma$ queue length bound even though this scenario does not closely match the analysis in Appendix C.1.

4.4 Multiple Bottlenecks

When subjected to a series of bottlenecks, IEC virtual links receive less share than cross-traffic passing through a single bottleneck, a.k.a., beat-down. However, it experiences substantially less

[2]labelled in Figure 10 as "Inverse scenaior w/ IEC + FIFO."
[3]labelled in Figure 10 as "T=ertt w/ IEC + FIFO."
[4]maximum queue length measured in the last 50 seconds of the simulation

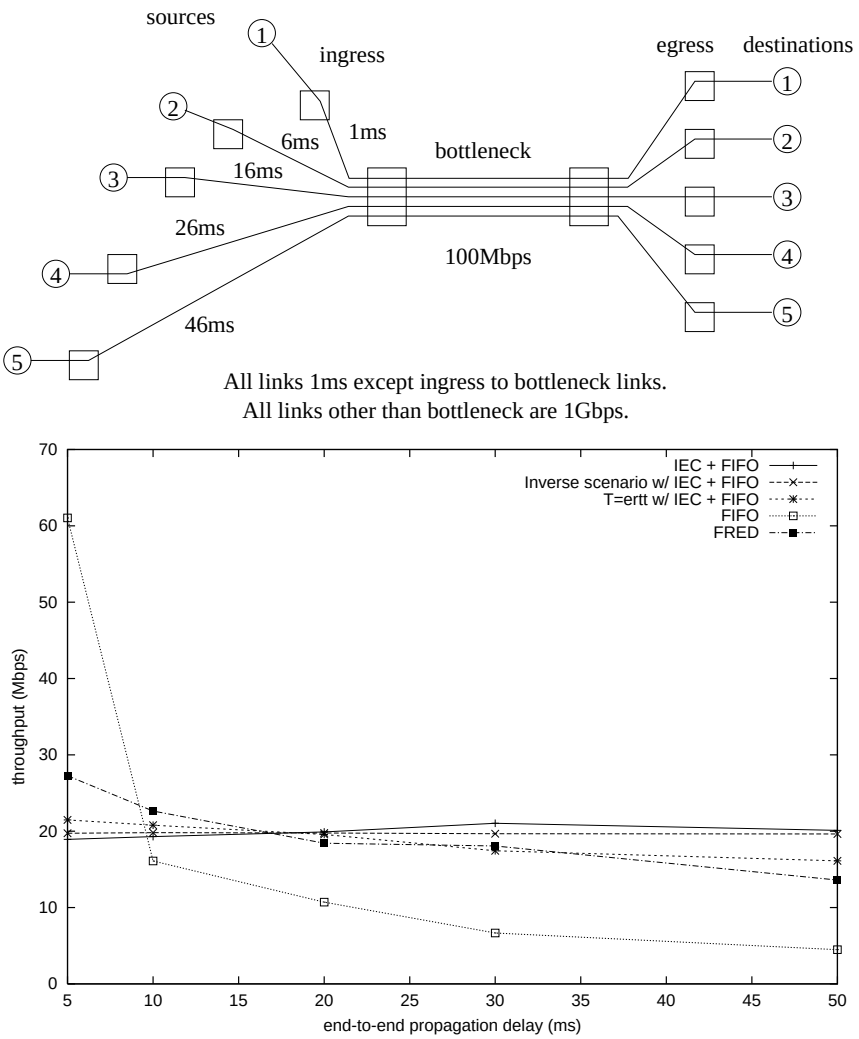
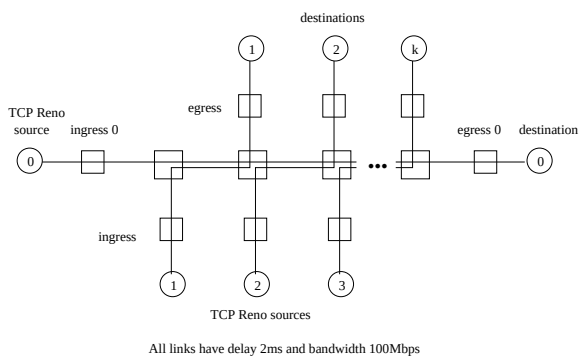
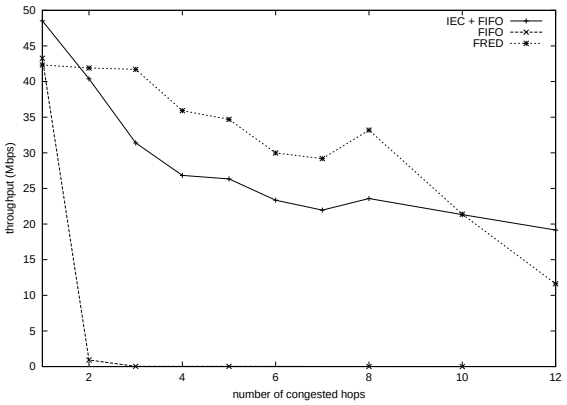


Figure 10: IEC distributes capacity evenly between TCP Reno connections whether delay is inside or outside the edge-to-edge control loops.

beat-down than non-edge-controlled TCP Reno connections subjected to a series 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.* IEC performs somewhat worse than FRED over most of the range of interest, but note that IEC requires no support from the interior.



(a) topology



Fairness of FIFO, FRED, and IEC+FIFO [a]

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

Figure 11: IEC experiences less beat-down than FIFO

5 Applications of Edge-to-edge Control

We present two applications of edge-to-edge control: distributed buffer management and an end-to-end low-loss best effort service.

5.1 Distributed Buffer Management

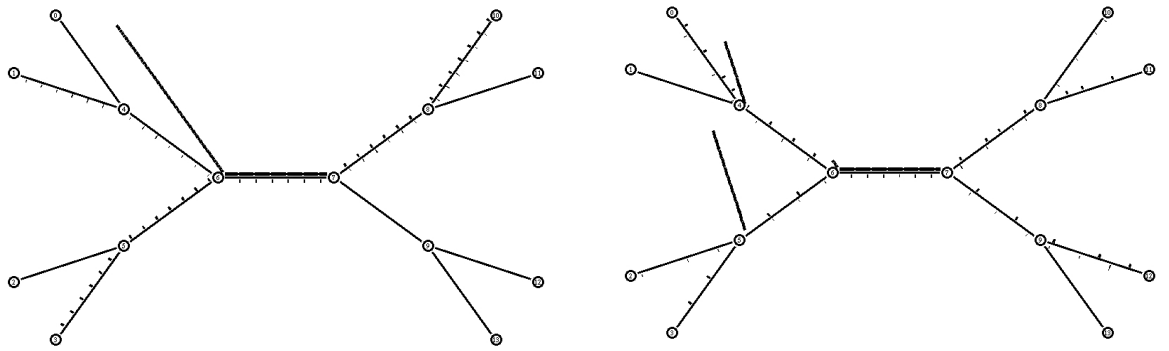
As already stated, edge-to-edge control can be used to distribute backlog across the edges (see Figure 12). This increases the effective number of buffers allowing more TCP connections to obtain large enough windows to survive loss without timing out. This reduces TCP's bias against tiny flows and thus improves fairness [13]. As demonstrated in Figure 13(a), using IEC to distribute the backlog dramatically reduces the coefficient of variation in goodput[5] 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 IEC, we can increase the effective number of buffers available to FRED by replacing FRED in the core with a FIFO and implementing FRED at the head of each virtual link (i.e., at each ingress edge). As expected, the coefficient of variation in goodput improves in some cases by as much as an order of magnitude.

5.2 Low-loss Best-Effort Service

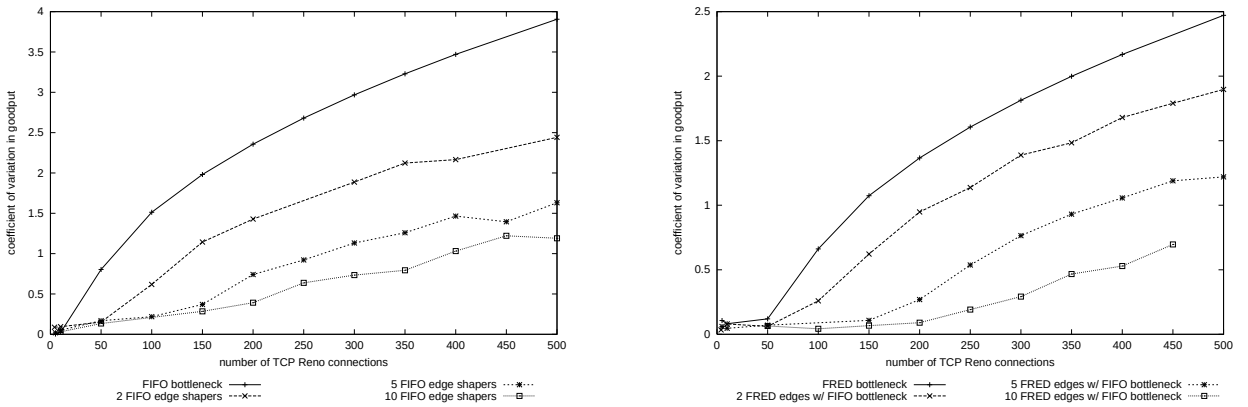
Recall that edge-to-edge control essentially breaks up core congestion and moves them to the edges. Given such behavior, we can place sophisticated control mechanisms at the edge thus avoiding inflicting such complexity on the network interior, and yet achieving superior performance. We demonstrate this capability by simulating the integration of IEC with TCPR (Packeteer's TCP Rate

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



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

Figure 12: Backlog moved from Bottleneck to Edge



(a) FIFO bottleneck versus IEC+FIFO edges [a] [b] (b) FRED bottleneck versus IEC+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

control[11]), i.e. (IEC+TCPR). As with IEC alone, neither IEC+TCPR does not require upgrading end-systems or interior routers provided interior routers support per-class queueing in order to separate edge-controlled and non-edge-controlled traffic. The buffer management integration is done only at the edge.

The IEC+TCPR is an interesting combination because it implements what is known as a "perfect callback", i.e. have sources send packets only when the network has capacity to accept

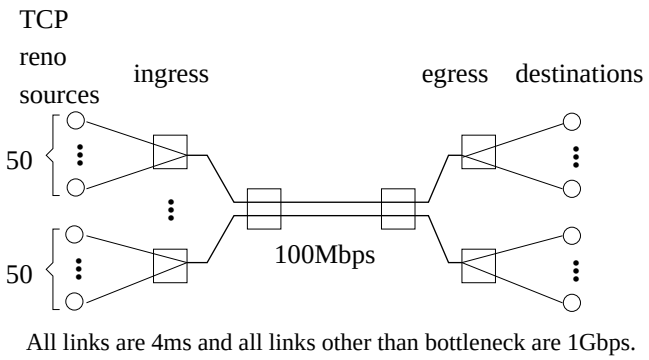


Figure 14: Topology used in comparing Edge Mechanisms

them. Such a perfect callback leads to a *negligible-loss, near-zero-timeout end-to-end TCP service.* In particular, this perfect callback manifests in two stages: first IEC pushes the congestion from the core to the edge and then TCPR pushes the congestion from the edge back to the source. To accomplish this the edge-to-edge virtual link ascertains the available capacity at the bottleneck and provides a (variable) rate-limit to the TCP rate controller (TCPR). The TCP rate controller [11] then converts the rate-limit to the appropriate TCP window size and stamps the window size in the receiver advertised window field in each acknowledgement heading back to the source. It is these aspects of TCPR which does not require packet loss to trigger TCP congestion control [11].

The performance of these options is illustrated in Figure 15 which has four rows. The first two rows illustrates TCP/IP performance when edge-to-edge control is not implemented, but FIFO (first row) or FRED (second row) is implemented at the bottleneck. The third row shows IEC+FRED (FRED at the edge) and the fourth row shows IEC+TCPR (TCPR at the edge). In the last two rows, the interior node implements FIFO. Note that the last two rows tradeoff an increased ingress edge queue (fourth column) for total performance in terms of other metrics.

The third row in Figure 15 shows that IEC+FRED provides better best-effort service than the first two rows, i.e., FIFO or FRED implemented in the core. A larger end-to-end delay (58.3 ms vs 39.4 ms) and ingress packet drops (3508 vs 0) is traded off to achieve superior bottleneck queuing and TCP service performance (i.e. lower retransmissions and timeouts). Notice that utilization

(third column) is not a differentiating metric in these simulations.

The last row in Figure 15 shows IEC+TCPR operating better for nearly every crucial metric. In particular, all the four latter column entries are zero, i.e., there is *zero packet loss, zero TCP retransmissions and zero TCP timeouts!* This is phenomenal service by any best-effort measure, especially for TCP which traditionally expects packet loss to signal congestion. Also note that the average end-to-end delays (fifth column) which includes transmission and queueing is 22.6 ms. In other words, the total *average queuing delay* at the edge and the core combined is a mere 2.2 ms, i.e. 90% lower than the equivalent value in the first row (canonical case). As a caveat this IEC+TCPR solution has a limited deployment space because the edges must have access to the TCP acknowledgement stream. However, this can be ideally done in a CPE-based VPN deployment.

In summary, our overlay architecture not only enables edge-based better-best-effort services, but also simplifies buffer management requirements in the network interior.

Ingress[a] Type	Bneck Type	Util-ization	Ingress[b] Queue	Bneck[c] Queue	Avg Delay[d]	Ingress Drops[e]	Bneck Drops	TCP RTX[f]	TCP RTO[g]
FIFO	FIFO	99.99%	small/1[h]	284.1/500	42.8ms	0	3682	3700	0
FIFO	FRED[i]	100.0%	small/1[h]	243.2/266	39.4ms	0	6389	7863	724
IEC+FRED	FIFO	99.00%	237.3/267	0.861/8	58.3ms	3508	0	4214	230
IEC+TCPR	**FIFO**	**99.06%**	**14.54/54**	**0.731/9**	**22.6ms**	***0***	***0***	***0***	***0***

[a] Statistics gathered for the last 20 seconds of a 30 second simulation

[b] Mean and max queue length in packets across both ingress edges respectively

[c] Mean and max bottleneck queue length in packets respectively

[d] Average end-to-end delay incl. propagation and transmission delays. Propagation delay is 20ms. Transmission delays total 0.4ms

[e] Sum drops from both ingress edges.

[f] Total TCP retransmissions

[g] Total TCP retransmission timeouts

[h] Ingress is not saturated and does not implement edge control. So we expect low queue lengths

[i] In all simulations we use FRED parameters as specified in section 4.5 of [13].

Figure 15: Edge mechanisms for controlling TCP: "FRED at the edge" and "TCPR at the edge"

5.2.1 Low-loss service under many TCP connections

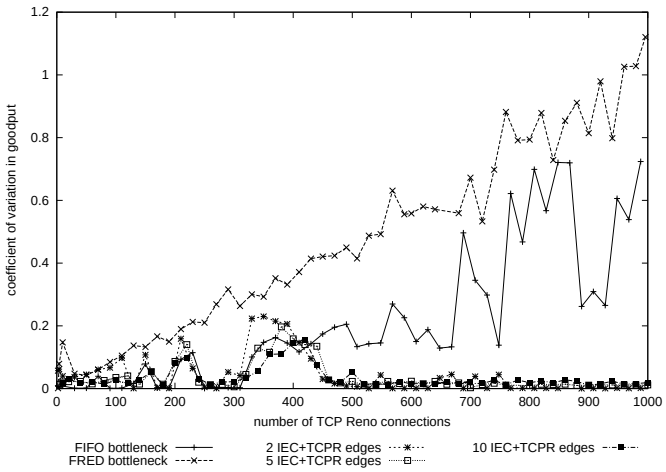
Using the same topology shown in figure 14, we vary the number of TCP Reno connections and the number of ingress edges. In all simulations, the TCP Reno connections are evenly distributed across the ingress edges. The results are graphed in Figure 16. Note in all of these simulations *IEC+TCPR achieves zero loss, zero retranmissions, and zero retransmission timeouts* in the steady state and the utilization is near 100%.

As noted in [15], with schemes like FIFO or FRED at the core, TCP's fairness begins to degrade significantly once the number of connections exceeds the number of packets in the bandwidth-RTT product (> 500). We show similar degradation for FIFO and FRED cases in our graph plotting coefficient of variation of goodput[6] (figure 16(a)). This occurs because each TCP connection must have at least a window size of four packets to survive a packet loss without incurring a timeout. When this is impossible the rate of timeouts increases (see figure 16(c)).

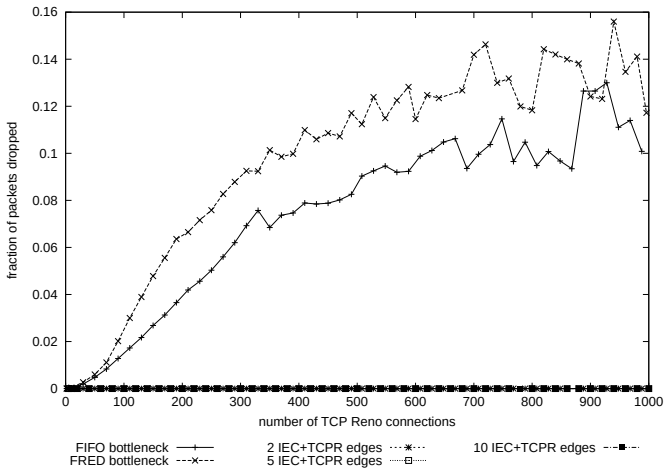
Since TCP rate control does not rely on loss to control TCP connections, it avoids incurring timeouts or unfairness by evenly reducing each TCP's advertised window size until all TCP connections send with window size 1. At this point, our implementation of TCP rate control does not reduce TCP send rate further. This is a modeling limitation of our simulation which is not reflective of Packeteer's implementation. Thus once the number of flows increases above the bandwidth-RTT product (> 500), our implementation of TCP rate control introduces a persistent backlog and the end-to-end delay begins to increase (see figure 16(c)).

As in the previous section, for a saturated bottleneck and a small number of connections, edge-to-edge control plus TCP rate control achieves an end-to-end delay that is comparable or better than FIFO. However as the number of virtual links increases, so does the delay. This is can at least partially be attributed to increasing bottleneck queue length as number of virtual links increases.

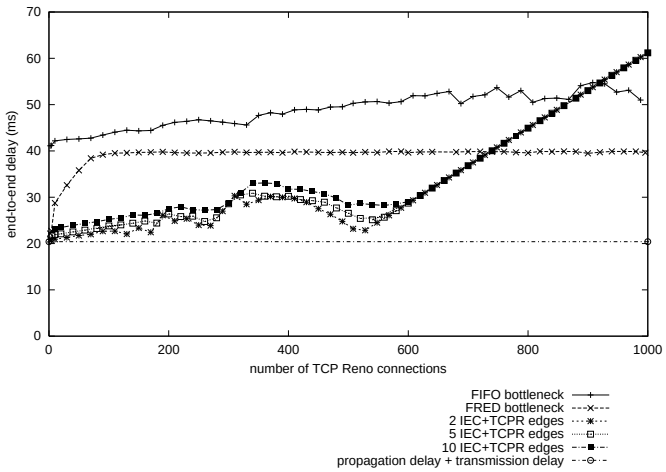
[6] Goodput is the number of useful payload bits transmitted through the bottleneck per second, i.e., not including retransmitted packets.



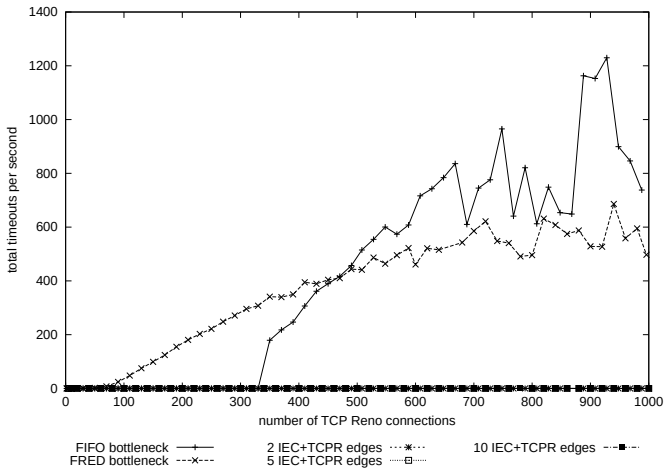
(a) Coefficient of variation in goodput vs flows



(b) Drop rate vs flows



(c) End-to-end delay vs flows



(d) Total timeouts per second vs flows

Figure 16: IEC plus TCP Rate Control under heavy loads (10Mbps bottleneck)

6 Incremental Deployment Strategies

Implementation and deployment flexibility is an important concern in modern protocol design. Questions of interest to ISPs and vendors in our work include:

- the applicability domain of the technology,
- the potential for incremental deployment and what technological aspect drives it.
- the value-proposition for ISP customers

Our overlay architecture is generic, which means that it can be applied to a variety of underlying networks such as: IP, ATM, frame relay, MPLS, X.25 or even over leased lines or SONET circuits. The goal would be to enable novel edge-based services on top of these networks. The list also shows that the technology can fit at either layer 2 or layer 3 in the protocol hierarchy. The architecture can be also leveraged to build customized services for a wide variety of Layer 4 (L4) and application traffic types. This allows a potentially large application domain from a technological viewpoint. Next we show that the architecture allows flexibility in terms of deployment strategies (for ISPs) and implementation options (for vendors).

Incremental deployment is possible on a customer-by-customer basis. Assume a TCD with a class set aside for our architecture. The deployment is then straighforward with virtual circuit based technologies like MPLS, ATM, frame-relay etc. In these cases, the new site-to-site virtual circuits can be mapped to the class set aside, and the edge-to-edge control can be operated over the VCs. With connectionless technologies like IP (diff-serv), there arises the problem of mapping end-to-end (micro-flows) to edge-to-edge virtual links (or loops). It can be solved for a private network scenario as follows. First exchange each customer's per-site IP address prefixes with other sites of the same customer. Since each site knows its own prefixes and the prefixes of other sites, micro-flows corresponding to the prefix combination will be mapped to site-to-site loops through simple prefix filtering. Policy-based architectures like COPS can be used for such prefix exchanges

and mapping establishment. In other words, the technology can fit as part of an IP-based VPN or SLA deployment initiative. Inter-domain and public network deployment issues are a topic for future study.

From an implementation (vendor) perspective, the technology can be implemented either in the data-plane (i.e. forwarding path of information flows) or in the control-plane (i.e. on management servers interfacing to data-plane components). A data plane solution could be a new "services box" or an integration of this technology with "access" or "edge" products. The latter solution avoids duplication of hardware platforms. The control-plane solution would interface to data-plane components through command-line interfaces (CLI). The performance and service capabilities of this alternative would depend upon the richness and responsiveness of the CLI.

Regarding the value-proposition, our core technology addresses the problem of congestion control scalability and provides a basis for edge-based bandwidth services. While ISPs have abundant core bandwidth there are still key niches (access links, aggregation points, tail circuits, international circuits and peering points) where the congestion problem exist and affect end-user performance. The goal of this technology is to overlay multiple loops around these points and move the congestion away to manageable points (edges) as illustrated in the paper. In the case of access links, an ISP might extends its TCD to place the edge in a Customer Premise Equipment (CPE) box.

From a services perspective, ISPs still face the problem of packaging their abundant bandwidth resources into high-value end-to-end services (SLAs). Current QoS and services technology (eg: diffserv [1] etc) requires fairly long development and deployment cycles. Our technology provides the foundation for edge-based services which can be deployed and updated rapidly. For example, ISPs could start from simple better best effort and L4 (eg: TCP) services and develop a larger base of services, possibly including dynamically contracted services, in the future (eg: [18]).

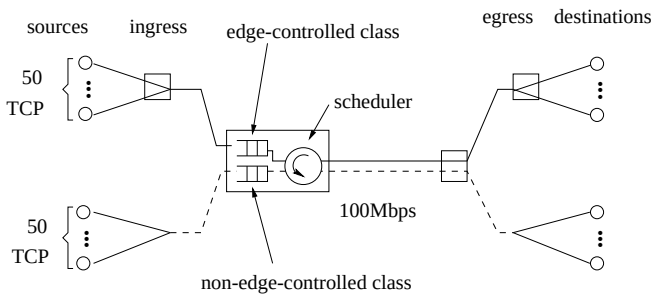


Figure 17: Class of Service Topology

6.1 Validation of the Basic Deployment Approach

This section provides a sample simulation result to validate the deployment approach of requiring a class-based core network where the edge-to-edge control architecture can co-exist with traditional TCP traffic. Figure 17 show a bottleneck with two classes (queues) serviced by a round robin scheduler. One class is for the edge-controlled traffic and the other for best effort. In this simulation our fair scheduler provides 50% of the bandwidth to each traffic class.

The results in figure 18 demonstrate that the distribution of capacity between the two classes is reasonably fair (46.8 Mbps edge-controlled versus 53.2 Mbps non-edge-controlled). The reason the split is not accurate is because the edge-to-edge control sometimes leads to zero queues and the capacity is captured by the uncontrolled class. During such a saturation period, the edge-to-edge control also backs off based upon measured egress rates.

In other words, the mapping of adaptively controlled traffic aggregates on top of a class leads to a slight beat down of the capacity allocated to the class. Simulations show that a 10% change in configuration parameters (53% vs 47% allocation) compensates for this situation. More importantly, observe that edge-to-edge controlled TCPs can co-exist (though isolated class-wise) with non-edge-to-edge controlled TCPs on a single bottleneck. Moreoever, when compared against the non-edge-controlled traffic, the edge-controlled class obtains *97% less excess delay*, *98.7% lower coefficient of variation in goodput* between TCP connections, as well as zero retransmissions and

Class[a]	Thruput (Mbps)	Ingress Q (avg/max pkts)	Bneck Q (avg/max pkts)	Avg Delay[b]	Excess Delay [c]	TCP RTX[d]	TCP RTO[e]	C.O.V. Goodput
IEC[f]+TCPR	46.8	11.4/55	0.38/5	22.31ms	1.91ms	0	0	0.00223
BE[g] TCP	53.2	n/a	45.3/500[h]	85.99ms	65.59ms	849	67	0.166

[a] Statistics for the last 20 seconds of a 30 second simulation.
[b] End-to-end delay
[c] Sum of queueing delays at ingress and bottleneck
[d] Total TCP retransmissions within class
[e] Total TCP retransmission timeouts within class
[f] Edge-controlled class using IEC
[g] Best effort, i.e., non-edge-controlled class
[h] The buffer is 500 packets

Figure 18: Class of Service Results

zero transmission timeouts, i.e., superior best effort performance.

7 Future Work

The edge-to-edge overlay control architecture is more than congestion control and is the basis for a future *service platform*. Another key contribution of this work is to achieve these benifits without requiring bottleneck support or requiring upgrades to end-systems. In this paper we have only demonstrated its usefulness for providing a better-best-effort service. We intend to broaden the service menu in future work. For example, we have initial results which suggest that we can provision the equivalent of a simple assured service (i.e. minimum bandwidth guarantee) completely edge-based using this architecture. In the future, schedulers at the ingress might distribute a virtual link's capacity according to application-level policy, or the schedulers might provide managed delay differentiation for interactive traffic. In the context of DiffServ, edge-control might inter-operate with the Assured Forwarding Per-Hop-Behavior [7] by performing all drop-preference at the ingress edge.

Future research will also have to address issues such as implementing edge-to-edge control and

end-to-end SLAs across multiple administrative domains. Also, to broaden the appeal of edge-to-edge control as a traffic management tool, edge-to-edge control might support hierarchies of overlaid virtual links. We will also investigate issues surrounding end-to-end deployment, though we do not see that as our core focus. Our current efforts include prototyping this architecture on a testbed and work with Internet2 on deployment and testing initiatives.

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] Brakmo, L., and Peterson, L. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communication 13*, 8 (October 1995), 1465–1480.

[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. 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.

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

[7] 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.

[8] Jacobson, V. Congestion avoidance and control. In *Proceedings of the SIGCOMM'88 Symposium* (August 1988), pp. 314–332.

[9] Jain, R. A delay based approach for congestion avoidance in interconnected hereogeneous computer networks. *Computer Communications Review, ACM SIGCOMM* (October 1989), 56–71.

[10] 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.

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

[12] Kung, H., Blackwell, T., and Chapman, A. Credit-based flow control for ATM networks: Credit update protocol, adaptive credit allocation, and statistical multiplexing. In *Proceedings of ACM SIGCOMM '94* (September 1994), pp. 101–114.

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

[14] 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).

[15] Morris, R. TCP behavior with many flows. In *Proceedings of ICNP '97* (October 1997).

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

[17] 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.

[18] Sinha, R., Kalyanaraman, S., and Ravichandran, T. Dynamic capacity contraction: A framework for congestion-sensitive pricing. Submitted to ICC'2000, 2000.

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

A Explicit Edge Control Algorithm

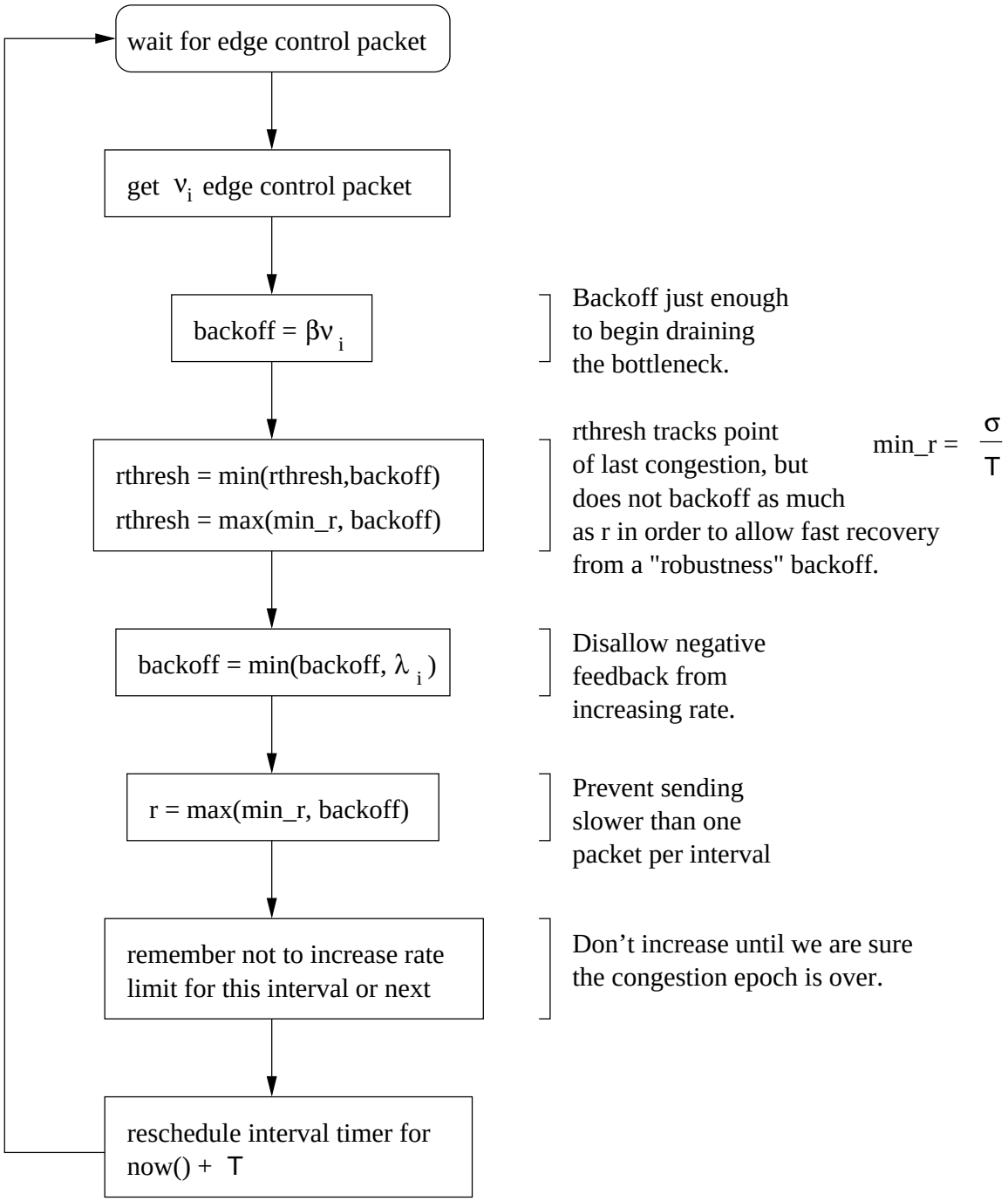


Figure 19: EEC Ingress Backoff without optional DFR or Robustness Backoff Algorithms

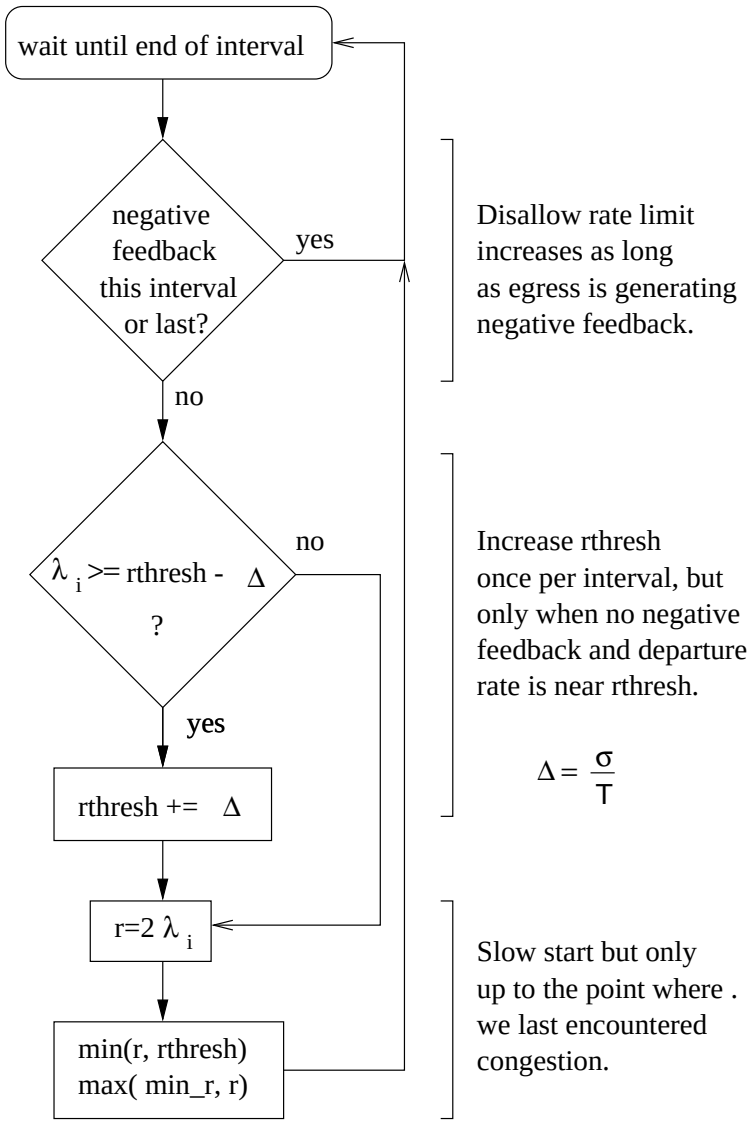


Figure 20: EEC Ingress End-of-Interval without DFR or Robustness Backoff Algorithms

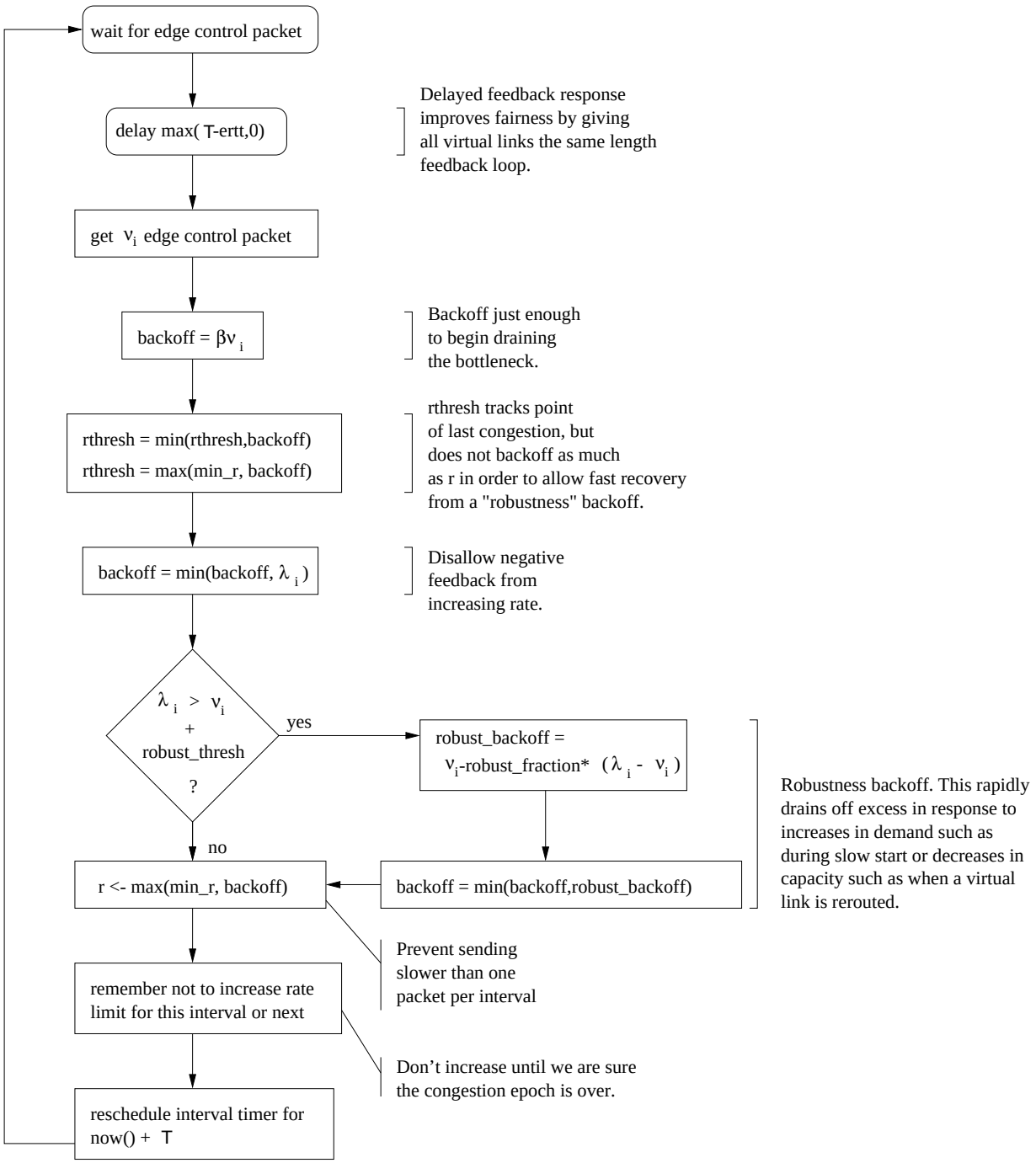


Figure 21: EEC Ingress with Delayed Feedback Response and Robustness Backoff

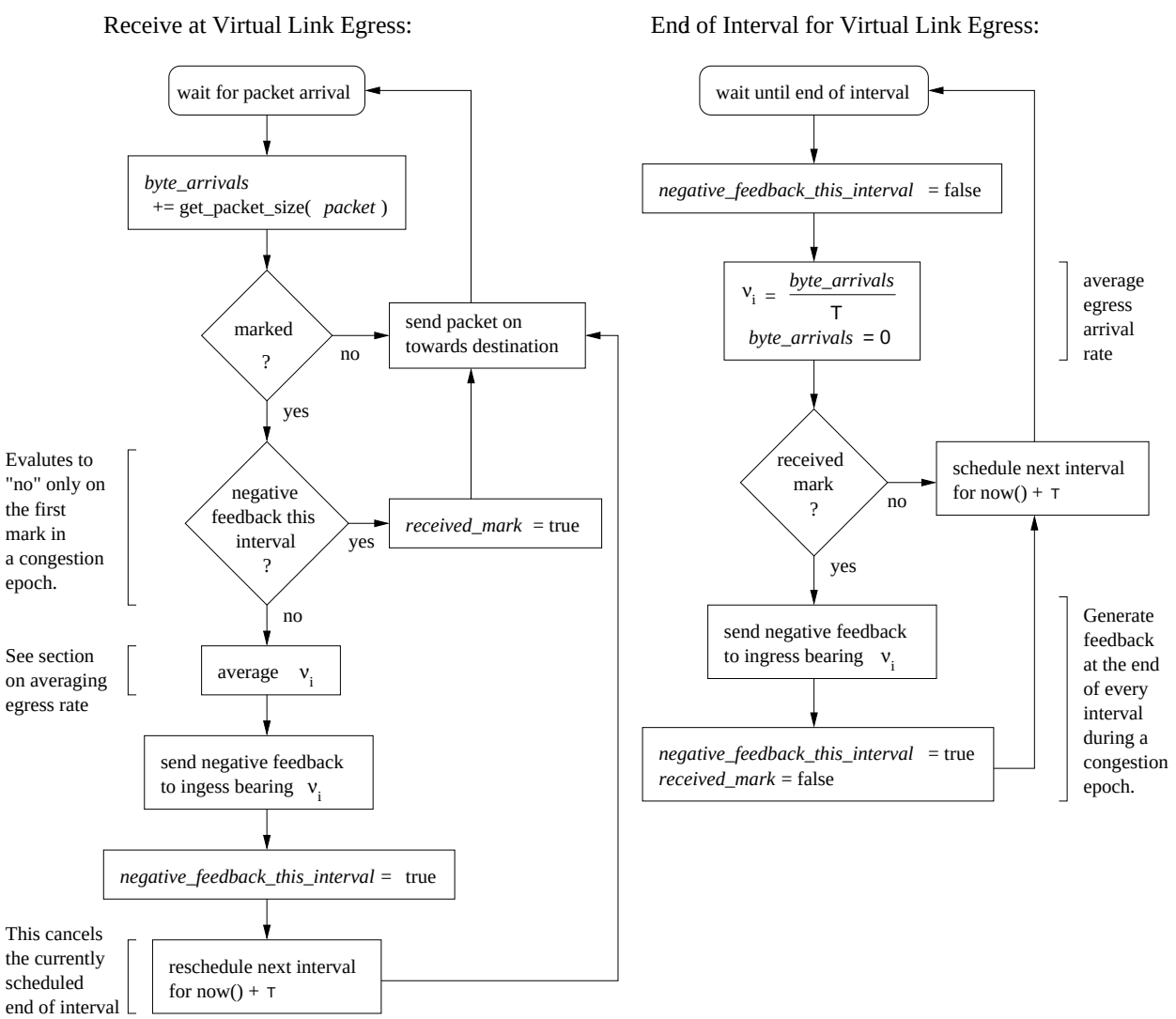


Figure 22: Egress Control Flow Diagram for EEC

A.1 Measuring Egress Arrival Rates in EEC

The EEC egress algorithm shown in figure 22 requires an algorithm for averaging the arrival rate. To do this, the egress counts the number of bytes arriving within the interval T. When a mark arrives, IEC averages the arrival rate of the previous interval with the arrival rate computed so far for the current interval as shown below:

$t = now() - begin_interval_time$

$\alpha = t/\mathsf{T}$

$\nu_i = (1 - \alpha)\nu_i + \alpha * byte_arrivals/t$

The algorithm above smoothes the average when the average is computed in the middle of an averaging interval. This occurs whenever a marked packet arrives (unless the marked packet just happens to arrive at the end of an interval).

A.2 Measuring Round-Trip Times in EEC

To measure edge-to-edge round trip times in the marking scheme, the ingress end of the virtual link could infrequently ping the egress end of the virtual link and use an exponentially weighted moving average between samples. Sharp changes in round-trip time such as when a virtual link changes its path through a domain may introduce substantial unfairness until the round-trip time updates. Our current simulations use a constant input parameter for the RTT value in the delay computation for the delayed feedback response mechanisms. Even with a constant, the edge-to-edge virtual links are remarkably fair. We leave determination of appropriate times between pings and the effects of route updates to future research. However, we expect route updates will have minimal impact on mean performance since route updates usually occur on much larger time-scales (e.g., minutes).

A.3 Robustness Backoff

When a virtual link slow starts or a route updates, the demand may suddenly exceed the available capacity. AIMD-ER handles small fluctuations in demand or capacity well, but AIMD-ER takes siginificant time to drain a large queue. To handle such scenarios, EEC and IEC can implement an optional *Robustness Backoff* algorithm. The Robustness Backoff algorithm adjusts the input rate to $\lambda_i = \nu_i - \text{robustness_fraction} * (\lambda_i - \nu_i)$ whenever the input rate λ_i exceeds ν_i by more than seen during normal steady-state oscillation. The robustness backoff algorithms can be seen in Figures 21 and 23.

A few initial experiments showed that if we used the Robustness Backoff to control backoff during steady state, then the virtual link would either overreact to slight fluctuations or underreact to large changes in demand or capacity. By waiting to trigger the robustness backoff until a threshold in the rate difference is exceeded, AIMD-ER and the Robustness Backoff mechanisms can be tuned separately.

B Implicit Edge Control (IEC) Algorithm

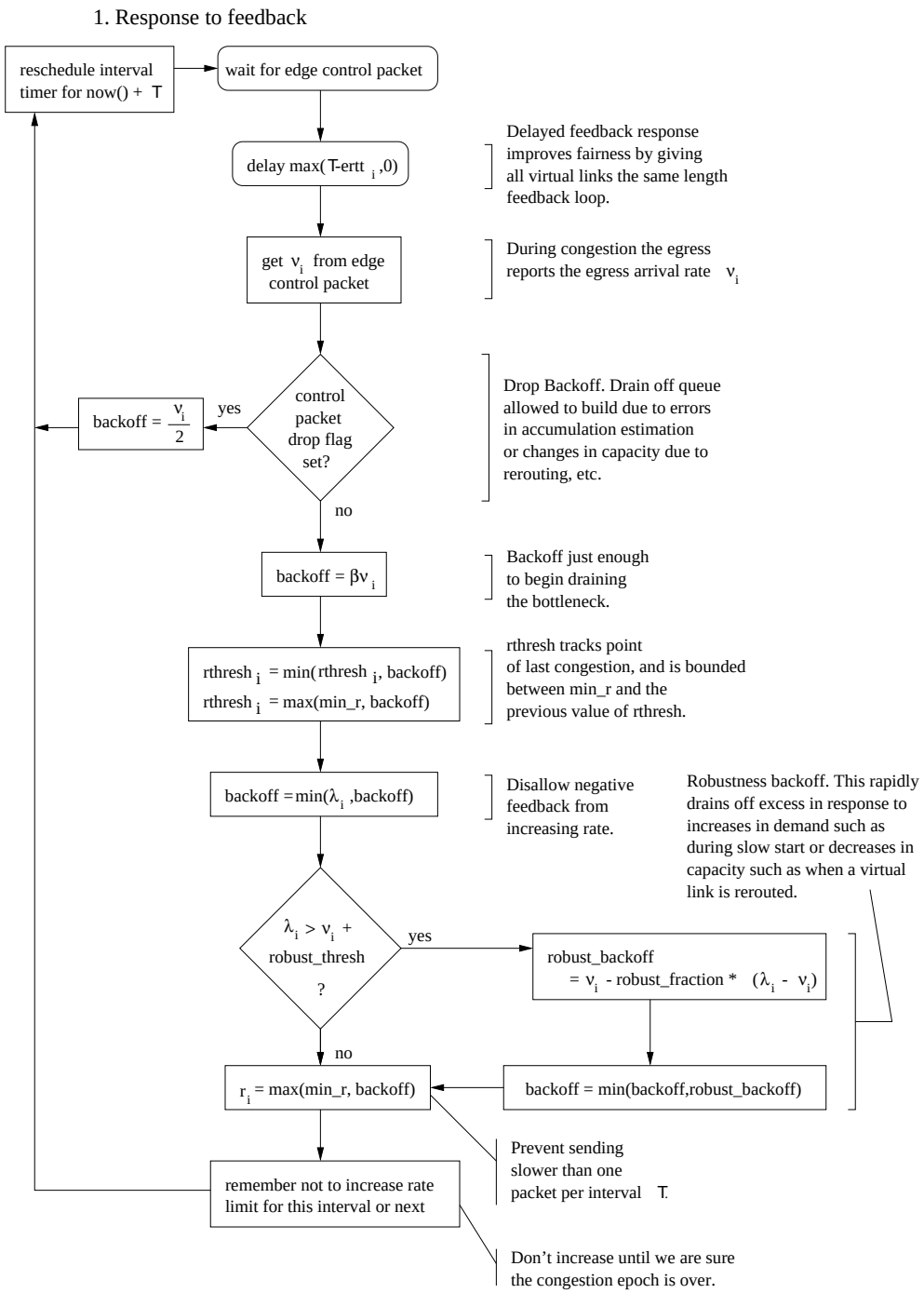


Figure 23: Ingress Backoff for Implicit Edge Control with Delayed Feedback Response, Drop Backoff, and Robustness Backoff algorithms

2. End of interval calculations

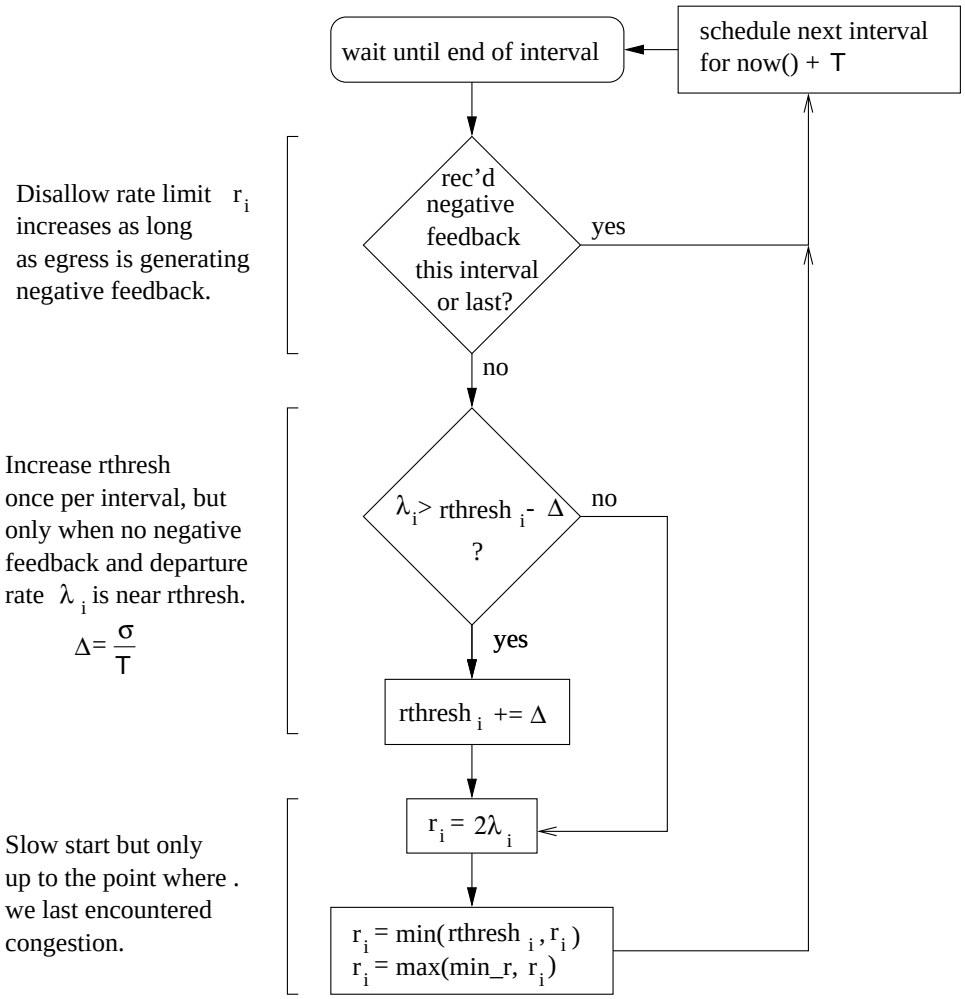


Figure 24: Ingress End-of-Interval for Implicit Edge Control

3. Insert forward control headers

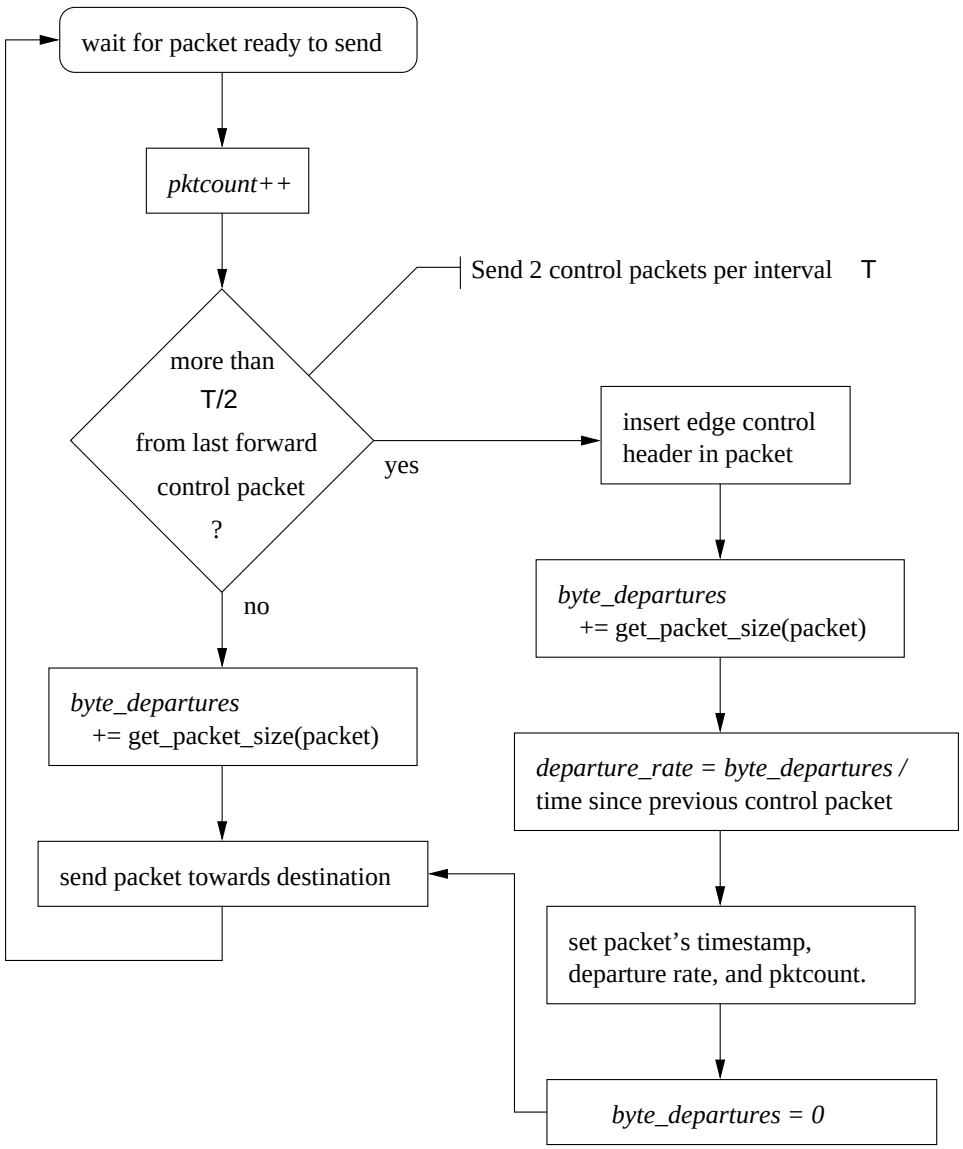


Figure 25: Ingress sending packets with Implicit Edge Control

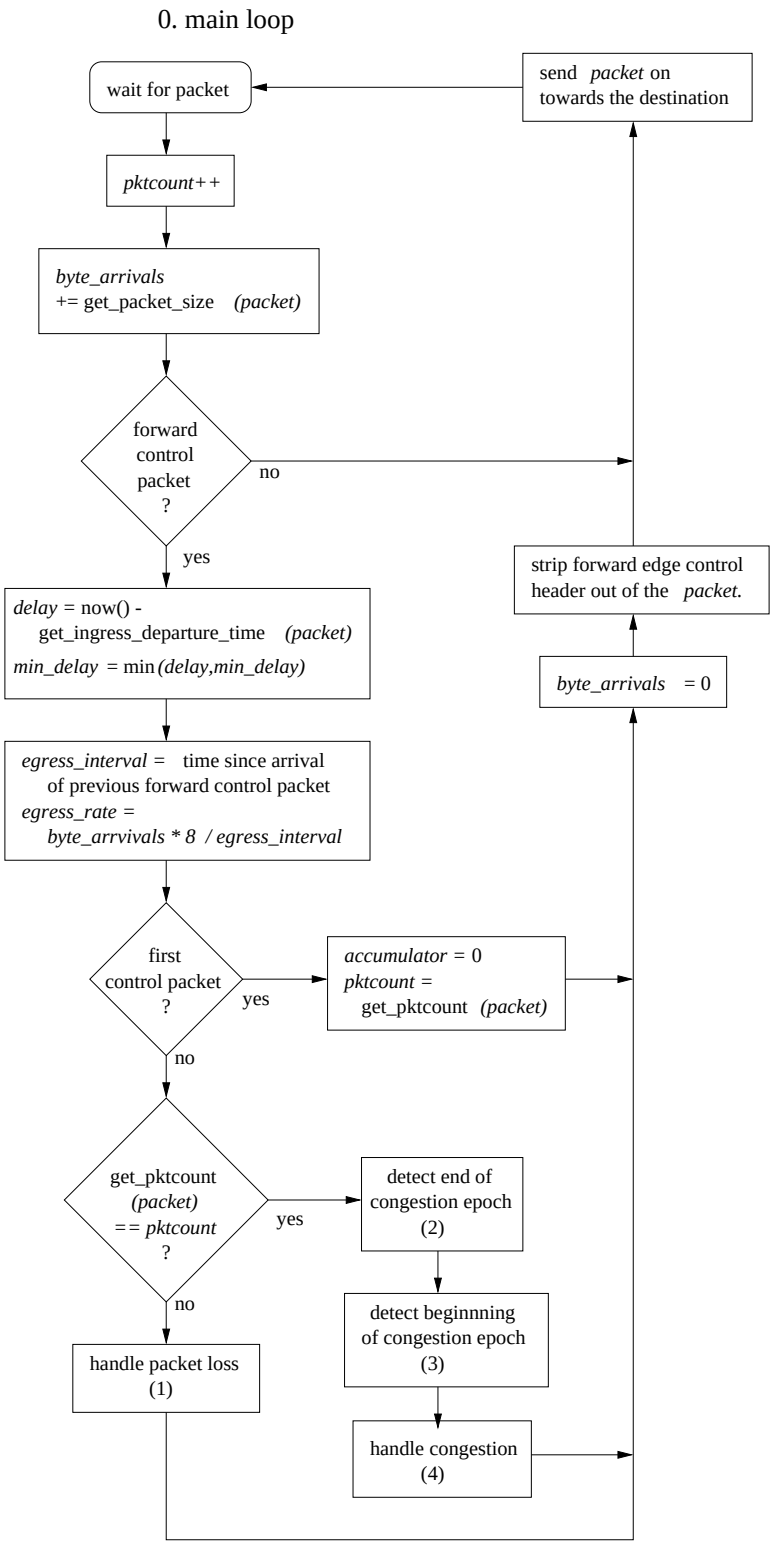


Figure 26: Main Egress Control Flow for Implicit Edge Control

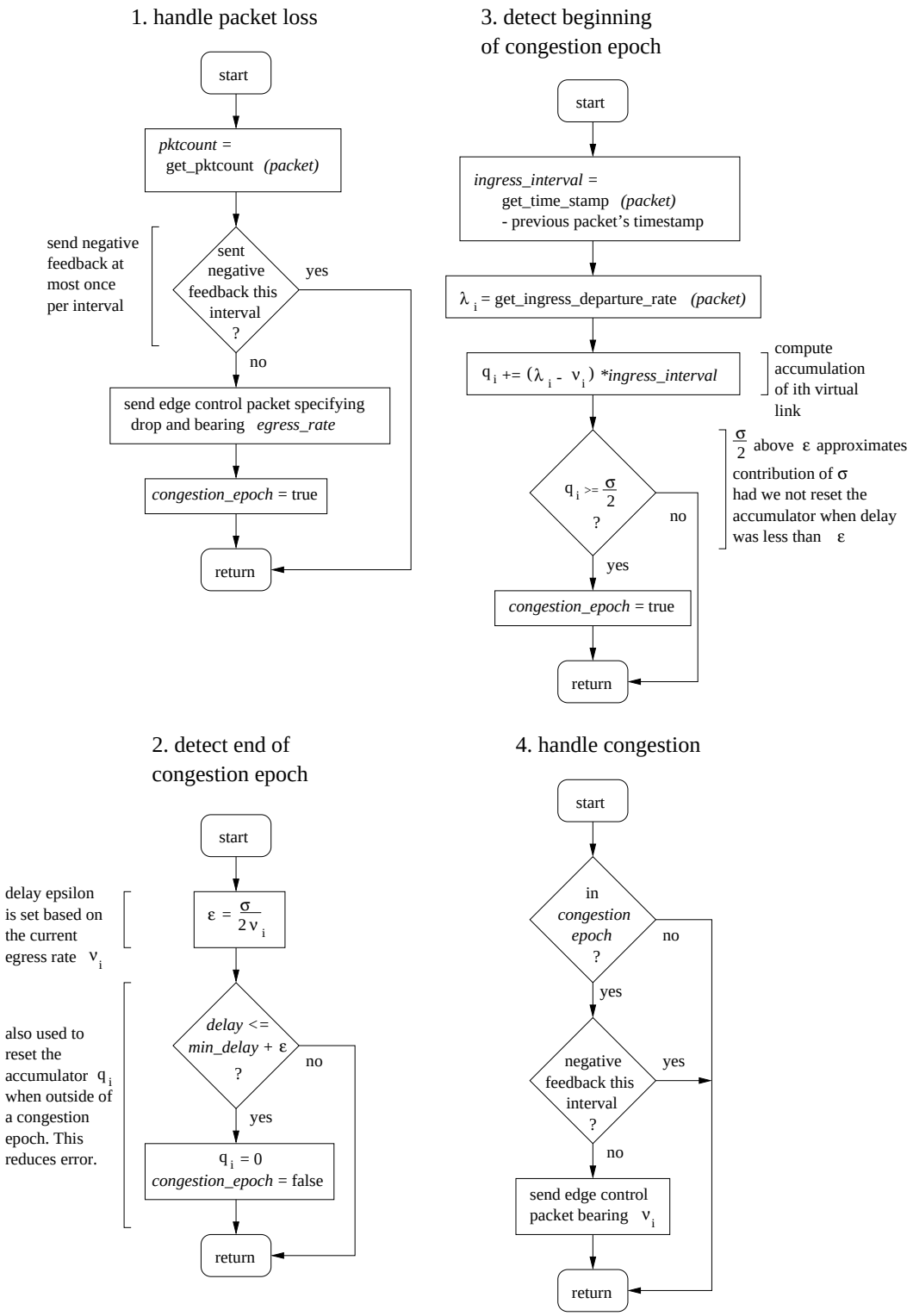


Figure 27: Egress Sub-diagrams for Implicit Edge Control

The IEC throttle control flow diagram in Figure 23 includes the robustness backoff and delayed feedback response mechanisms. These are optional mechanisms.

B.1 Measuring Arrival Rate in IEC

IEC does not rely on marks from the core and thus does not have to worry about congestion epochs beginning at any time. Instead congestion epochs can only begin when a control packet arrives from the ingress. Since forward control packets arrive every half interval, the rate averaging computation times are well-spaced out. As a result, IEC averages the arrival rate simply by dividing the *byte_arrivals* counter by the time between control packets.

B.2 Measuring Round-Trip Times in IEC

As with the EEC simulations, we used a constant input parameter for the RTT used in the delay computation for the delayed feedback response mechanism. In reality some mechanism would be necessary for measuring the round-trip time. The pinging mechanism introduced for EEC would also work for IEC.

B.3 Keep-alives

So far we have assumed that the reverse feedback channel is reliable. To handle the unreliability of the reverse channel, after an absence of feedback the egress sends a keep-alive packet. If the ingress does not receive a keep-alive packet it times out and reduces the rate by half and continues to reduce by one half each interval until either a keep-alive or negative feedback arrives. The alternative is to have an egress-driven bipolar scheme where the ingress makes rate increases only upon receiving "no-congestion" control packets from the egress. Again, a timeout is used here to reduce the sender rate if no control packets are received for a long time. The benefit of "keep-alives" is that the overhead can be tuned by setting the maximum time without feedback before a keep-alive is generated. We have implemented both strategies, but both EEC and IEC simulations

in this paper use keep-alives.

B.4 Accumulation Approximation Details

In Section 3.2.1 we stated that the queue length at time t obeys:

$$\begin{aligned} q_i(t) &= q_i(t-\tau) + (\overline{\lambda}_i[t-\tau, t] - \overline{\nu}_i[u-\tau, u])\tau \\ u &= t + \text{fixed delays} \end{aligned} \tag{5}$$

By this equation, the contribution of virtual links with a smaller share of the capacity will be proporationally smaller. As a result, larger virtual links will reach the contribution threshold σ earlier. This provides a push towards fairness not present in EEC.

We also stated that the averaging interval for $\overline{\nu}_i$ is the time between the arrivals of control packets at the egress. Unfortunately control packets are affected by queuing delays (see Figure 28). Thus we denote jth *egress averaging interval* as $[u_{j-1}, u_j]$. The duration of the egress averaging interval can be shorter or longer than the averaging interval $[u-\tau, u]$ as shown in Equation 5. As a result we can only approximate $q_i(t)$. We denote this approximation as $\tilde{q}_i(t)$ as shown below:

$$\tilde{q}_i(t_j) = \tilde{q}_i(t_{j-1}) + (\overline{\lambda}_i[t_{j-1}, t_j] - \overline{\nu}_i[u_{j-1}, u_j])(t_j - t_{j-1}) \tag{6}$$

When all N virtual links have σ burstiness then the queue length is $N\sigma$. As already stated this is the minimum mark threshold for EEC. However, IEC only loosely models EEC. In fact prior to convergence near fairness IEC virtual links can impose backlogs significantly larger than EEC's steady-state queue length bound. However, as I will show below, the occassional failure of some virtual links to backoff immediately once the queue length reaches $N\sigma$ actually pushes the system towards fairness. In section C.2 we provide a steady-state queue length bound for IEC that takes into account the more obvious differences between IEC and EEC.

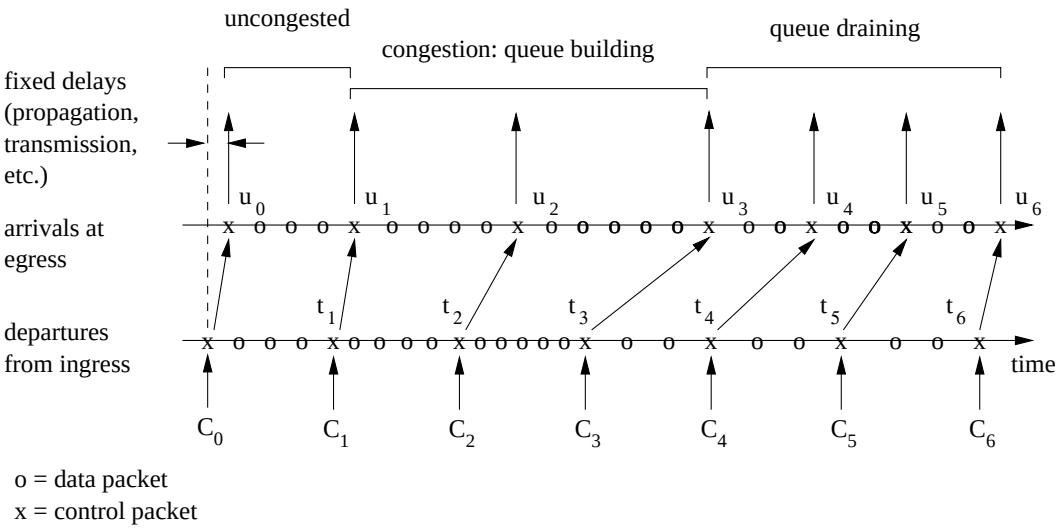


Figure 28: Bottleneck Queue Time Diagram

Using a fluid model over a single bottleneck topology and assuming synchronized intervals, we can show that the accumulation method punishes VL's with larger arrival rates. Furthermore, the fluid rates only change at the beginning of each interval. The last constraint is necessary to move from equation 7 to equation 8.

$$\begin{aligned}
\nu_i(u) &= \frac{\lambda_i(u-\text{delay})}{\lambda(u-\text{delay})}\mu \\
\overline{\nu}_i[u_{j-1},u_j] &= \frac{1}{u_j-u_{j-1}}\int_{u_{j-1}}^{u_j}\nu_i(\mathsf{t})d\mathsf{t} = \frac{1}{t_j-t_{j-1}}\int_{t_{j-1}}^{t_j}\frac{\lambda_i(\mathsf{t})}{\lambda(\mathsf{t})}\mu d\mathsf{t} \qquad (7)\\
\overline{\nu}_i[u_{j-1},u_j] &= \frac{\overline{\lambda}_i[t_{j-1},t_j]}{\overline{\lambda}[t_{j-1},t_j]}\mu \qquad (8)
\end{aligned}$$

We can then substitute equation 8 into equation 6 to obtain:

$$\begin{aligned}
\tilde{q}_i(t_j) &= \tilde{q}_i(t_{j-1}) + (\overline{\lambda}_i[t_{j-1},t_j] - \frac{\overline{\lambda}_i[t_{j-1},t_j]}{\overline{\lambda}[t_{j-1},t_j]}\mu)(t_j-t_{j-1}) \\
\tilde{q}_i(t_j) &= \tilde{q}_i(t_{j-1}) + \overline{\lambda}_i[t_{j-1},t_j]\left(1-\frac{\mu}{\overline{\lambda}[t_{j-1},t_j]}\right)(t_j-t_{j-1}) \qquad (9)\\
& \qquad (10)
\end{aligned}$$

In equation 9 the change in the accumulation is proportional to $\overline{\lambda}_i$, thus a larger VL's accumulator increases proportionally faster than a smaller VL's. Smaller VL's can continue to increase and capture capacity even while larger VL's have backed off.

Additionally, the accumulation estimate may be affected by limitations in clock resolution and clock skew. To limit the effects of all sources of error in the approximation, we reset the approximation whenever the delay on a control packet is within ϵ of the minimum delay. Since minimum delay is used to detect the end of a congestion epoch, $\tilde{q}_i$ is reset at least once per congestion epoch.

The effectiveness of these mechanisms is dependent on an appropriate setting of ϵ. If ϵ is too small then not all virtual links will detect the end of a congestion epoch before the system enters the next congestion epoch. If ϵ is too large then we cannot achieve the queue length bound of EEC. I empirically determined that $\epsilon = \frac{\sigma}{2\nu_i}$ works well. This is the delay incurred by half a maximum burst transmitted at the virtual link's current share of the bottleneck capacity.

Because we reset the approximate accumulation until the delay exceeds $\frac{\sigma}{2\nu_i}$, we adjust the accumulator threshold down to $\frac{\sigma}{2}$. The sum of half a maximum burst worth of delay and a half a maximum burst worth of queue length contribution is equivalent to using a queue length contribution of σ without the frequent accumulator resetting.

B.5 Clock Skew, Rerouting, and Drop Detection

Similar to TCP-Vegas[2], IEC relies on the measurement of the minimum delay. Thus we share some of the same problems as TCP-Vegas. Namely 1) when multiple sources become active in succession the latter sources may measure an incorrect minimum delay, and 2) when routes change the propagation delay may increase requiring a new minimum delay[14]. A succession of sources can lead to latter sources grabbing a disproportionately large share of the bottleneck, persistently long queues, and possible drops. In the latter scenario, the source confuses increased propagation delay with congestion and may reduce to a near zero share of the bottleneck.

If a series of virtual links become active one after another, the IEC suffers the same damage

as TCP Vegas. However, our system tends to correct the problem with time. Once the end of a congestion epoch is detected, our scheme continues draining the queue for at least one interval longer than required to reach $min_delay + \epsilon$. Provided ϵ is smaller than the delay incurred by the amount drained in the extra interval $(1-\beta)\nu_i \mathsf{T}/\nu_i = (1-\beta)\mathsf{T}$, the minimum measured delay will decrease with each congestion epoch until the system settles near zero queue delay. If ϵ sometimes becomes too large when set to $\epsilon = \frac{\sigma}{2\nu_i}$, we can increase the interval T or decrease β to solve the problem. The cost of the prior being decreased responsiveness and the latter being decreased utilization.

When a virtual link is rerouted the propagation delay may suddenly change. If the propagation delay decreases then the rerouted virtual link(s) have overly large measured minimum delays. This situation is solved by the same mechanism that solves the scenario where many virtual links begin in succession. Unfortunately the mechanisms stated so far cannot tell the difference between a congestion and an increase in propagation delay. [14] suggests a solution for TCP Vegas of keeping the last K delay samples. If the minimum in the last K is sufficiently larger than the measured minimum delay then use the minimum delay in the window. We have not tried this method yet.

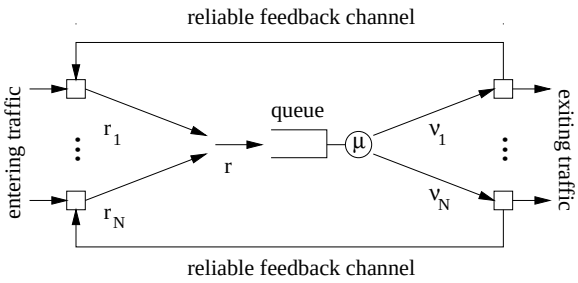
Unlike TCP Vegas, we have to face the problem of clock skew. TCP Vegas gets around this problem because it uses round-trip times rather than one-way delays and thus relies on only one clock. We shied away from using round-trip times only because one-way delays can be measured by piggy-backing timestamps on existing packets whereas round-trip times would require the egress to frequently send new packets to the ingress even when there is no congestion. In IEC, we measure the one-way delay twice in an interval without introducing additional packets. In an RTT scheme, we would have to send two new packets per interval per virtual link. If it turns out that clock skew causes problems over the time-spans that we are interested then we will consider using round-trip times.

In the event that all else fails, the virtual link detects loss and backs off by one-half from the egress rate. Such a drastic backoff often corrects problems introduced by a rapid succession of

activating virtual links or a decrease in propagation delay due to rerouting. To detect loss, both ingress and egress maintain a count of the number of packets received and the ingress puts this packet count in each control packet. When there is a mismatch, the egress indicates loss to the ingress by sending negative feedback with a loss flag.

C Analysis

C.1 Steady-State Queue Length Bound for EEC



$r(t)$	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 29: Topology, Constants, and Variables used in Analysis

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 29. 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
- all feedback passes through uncongested, reliable channels
- 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 has a leaky bucket obeying inequality 1. With N flows this inequality becomes:

$$I[t, t+\tau] \leq \int_t^{t+\tau} r(T)dT + N\sigma \tag{11}$$

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:

$$r(t) \leq Na(t - t_\mu) + \mu \tag{12}$$

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

$$q(t+\tau) = q(t) + I[t, t+\tau] - \mu\tau \tag{13}$$

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 $t = 1$.

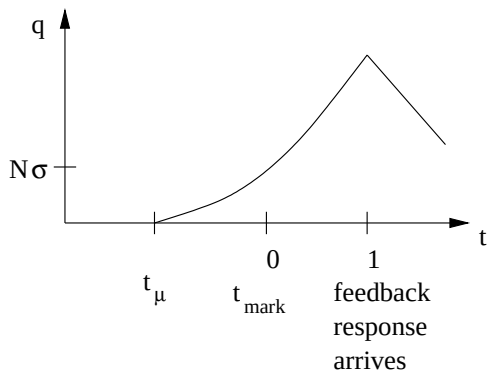


Figure 30: Queue length versus time

$$q(1) = N\sigma + I[0, 1] - \mu \leq N\sigma + (\frac{Na}{2} + r(0) + N\sigma) - \mu \tag{14}$$

All terms appearing on the right-hand-side of inequality (14) 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,

$$\int_{t_\mu}^{0} Na(T - t_\mu)dT = N\sigma, \quad t_\mu = -\sqrt{\frac{2\sigma}{a}}$$
$$r(0) = N\sqrt{2\sigma a} + \mu \tag{15}$$

Combining equation 15 and inequality 14 yields:

$$q(1) \leq \frac{Na}{2} + N\sqrt{2a\sigma} + 2N\sigma = \left(\frac{5}{2} + \sqrt{2}\right)N\sigma \tag{16}$$
$$\text{max q} \leq q(1) < 4N\sigma \tag{17}$$

C.2 Steady-State Queue Length Bound for IEC

With a few additional constraints we now demonstrate a hueristic steady-state queue length bound for IEC assuming IEC has already converged on fairness. We modify the model used in section C.1 as follows:

- The bottleneck does not mark.
- Edge control information is introduced into the fluid flow twice per interval T.
- All edges have synchronized intervals.

Note that all VL's still increase their rate limits at most continuously with acceleration a.

Once the system reaches fairness, all virtual links approach the same queue length contribution estimate:

$$\tilde{q_j}(t_j) = \frac{q(t_j)}{N} = q_i(t_{j-1}) + (\overline{\lambda_i}[t_{j-i}, t_j] - \frac{\mu}{N})(t_j - t_{j-1}) \tag{18}$$

Thus once the system converges on fairness, $\tilde{q}_i(t_j) \geq \sigma$ implies $q(t_j) \geq N\sigma$. As stated above this is the minimum mark threshold for EEC. However, IEC might not detect the queue length as quickly as EEC since control packets are sent only twice per interval. If we assume that the interval length is much larger than the queue delay imposed by $N\sigma$ packets then the detection is delayed by at most 1/2 of an interval (due to control packets being sent twice per T). With this assumption we can also use the exact same derivation as used for EEC in appendix C.1 up to inequality 14. Here inequality 14 becomes

$$q\left(\frac{3}{2}\right) = N\sigma + I\left[0, \frac{3}{2}\right] - \mu \leq N\sigma + \left(\frac{9Na}{8} + \frac{3}{2}r(0) + N\sigma\right) - \mu \tag{19}$$

Combining inequality 19 with appendix C.1 equation 15 and the definition $a = \frac{\sigma}{\mathsf{T}^2} = \sigma$ yields:

$$q(1.5) \leq \frac{9Na}{8} + \frac{3}{2}N\sqrt{2a\sigma} + 2N\sigma = \left(2 + \frac{3\sqrt{2}}{2} + \frac{9}{8}\right)N\sigma = 5.246...N\sigma < 6N\sigma \tag{20}$$

I have seen simulations where IEC reaches $5N\sigma$, but I have never seen the $6N\sigma$ bound violated once the system has converged near fairness.