Edge-To-Edge Traffic Control: A New Overlay Congestion Control Architecture for the Internet

Paper 188 (20 pages)

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 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 quickly deployed.

1 Introduction

We propose a new overlay architecture for edge-to-edge control where domain edges set up control loops with other edges and regulate aggregate traffic passing between each edge-pair, creating what we call edge-to-edge *virtual links*. These loops are overlaid at the network (IP) layer and can control both TCP and non-TCP traffic. The operation of the overlay 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 and 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 buffer management. 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 interior 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.

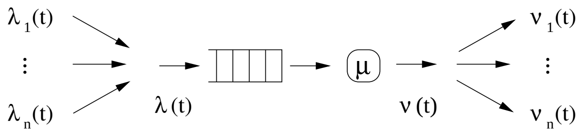


Figure 1: Traffic Control Model

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 [3]. 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 A.

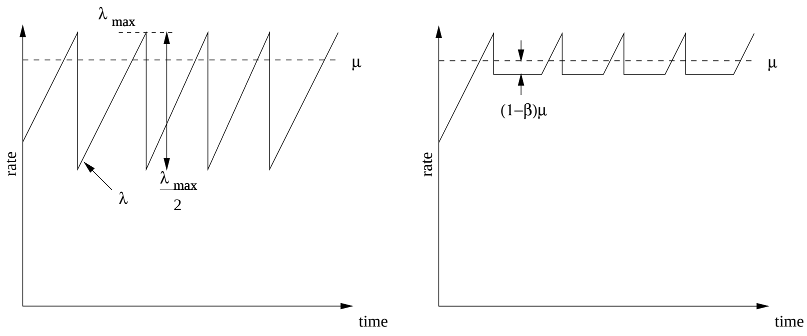


Figure 2: Steady-State Dynamics

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.

The second scheme does not involve support from interior routers. To detect the beginning of a congestion epoch, 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. The observation that output rates could be used as a basis of congestion response was first used in the design of

NETBLT[4]. 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 [14] and TCP Vegas [1], 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 [14]).

- 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 [13], DECbit [14]) and explicit rate schemes (ATM ABR) [15][9]. 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 [11] and the conservation-of-packets principle used in TCP [7]. 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 due to Jain [8] and the TCP-Vegas work [1]. We make a key observation that one-way delay cannot be used to indicate the *beginning* of a congestion epoch. This is because there is no reliable reference to compare one-way delay estimates against in this context. 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.

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 existence of *traffic control domains*, which is a set of networks where consistent policies are applied by a single administrative entity. 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).

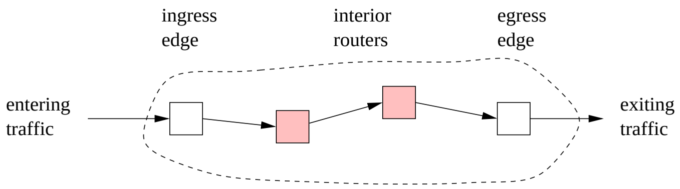


Figure 3: Traffic Control Domain

We aggregate traffic between any given ingress-egress edge pair onto an edge-to-edge feedback-controlled *virtual link*. The mechanisms for scheduling onto the virtual link are orthogonal to edge-to-edge control. Recently a edge-to-edge TCP trunking method was proposed by Chapman and Kung [2] where they propose the use of TCP-type AIMD control policies. Our use of newer control policies allows earlier congestion response and hence a wider application range.

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.

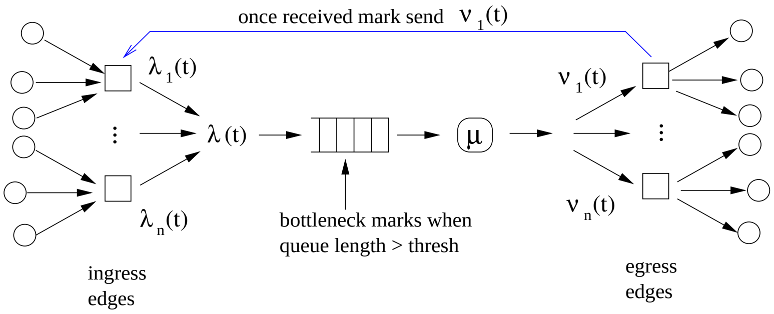


Figure 4: Ingress aggregates end-to-end flows into virtual links. When queue length exceeds threshold, all packets are marked. Upon receiving mark, egress generates negative feedback.

3.1.1 Scheme Details: Rate Adjustment Policies

The ingress node uses a leaky bucket rate shaper[5], whose rate limit (r) can be varied dynamically based upon feedback. The amount of traffic I entering the network over any time interval $[t, t+\tau]$ 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 can be set as $N\sigma$ 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 5.

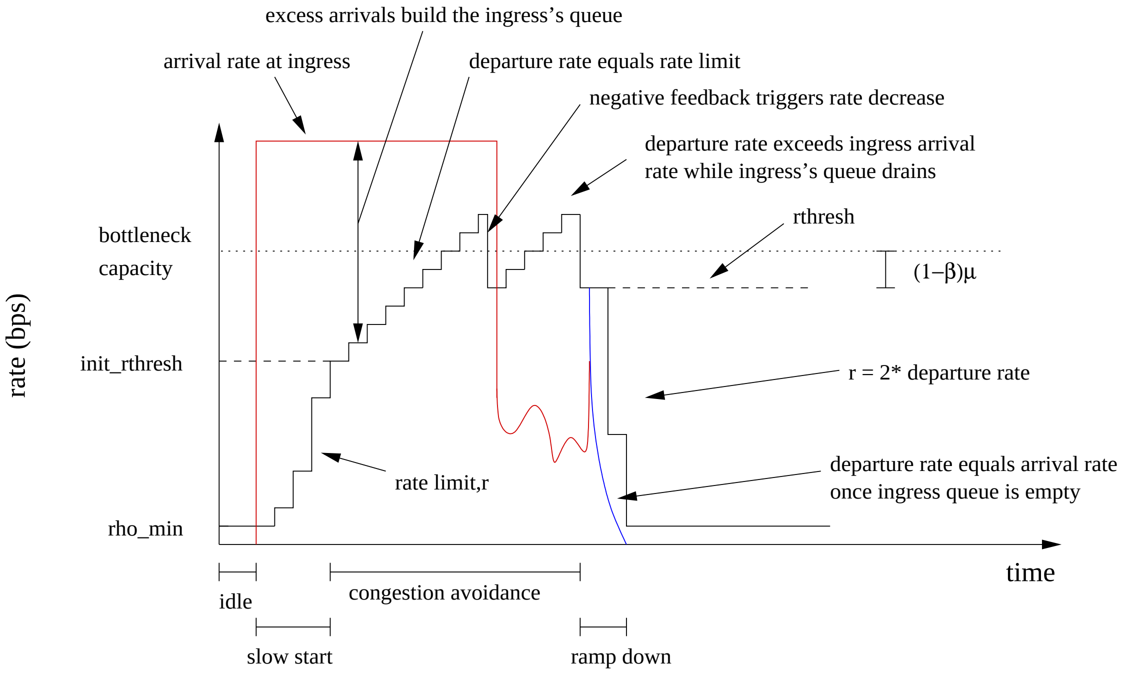


Figure 5: Detailed Dynamics of Edge-to-edge Rate Regulation

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 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 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 delay.

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.

Lastly, to quickly adjust to sharp changes in demand or capacity, the ingress backs off by $\mu_i/2$ when packet loss occurs.

3.2 Edge-to-edge Control Without Interior Support

Our second scheme detects congestion epochs without support from the interior routers, and is called *Implicit Edge Control (IEC)*. These new procedures are the first step in increasing the deployment space for our architecture. Besides the procedures and control packets required for congestion epoch detection, the IEC scheme is identical to the first scheme, Explicit Edge Control (EEC).

3.2.1 Detecting Congestion Epochs without Interior Support

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 accumulation q_i can be calculated using the following observation. Assume a sufficiently large interval T_c. If the average input rate during this period T_c is $\overline{\lambda}_i$, and the average output rate is $\overline{\nu}_i$, the accumulation caused by this flow during the period T_c is $(\overline{\lambda}_i - \overline{\nu}_i) \times T_c$.

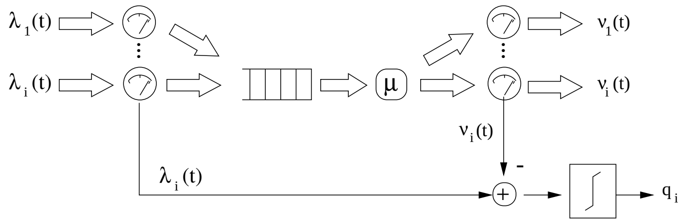


Figure 6: Measuring Virtual Link's Contribution to Queue Length

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.

The ingress node sends two control packets in each interval T (but no faster than the real data rate). T_c is the interdeparture 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: $(\overline{\lambda}_i - \overline{\nu}_i) \times T_c$ 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.

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.

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. Other measurement and robustness issues are discussed in our technical report [OMITTED].

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 7. A full suite of simulation results is presented in our technical report [OMITTED].

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[c]	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, 4.3, and 5.2.

[b]rtp=round-trip propagation delay

[c]applies only to EEC

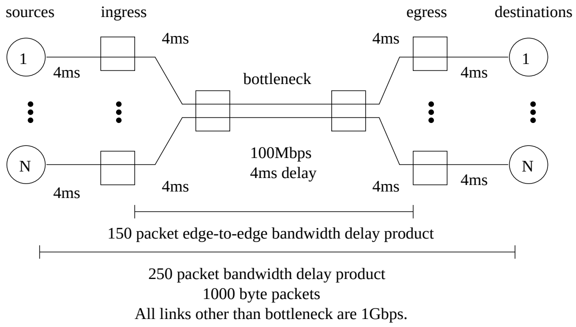


Figure 7: Simulation parameters and Single Bottleneck Topology

4.1 Single bottleneck

These simulations demonstrate the EEC and IEC schemes controlling persistent UDP flows[1]. Figure 8 shows that both EEC and IEC stay within the analytically-derived steady-state queue length bound $4N\sigma$ (derived in Appendix A), have similar mean bottleneck queue lengths, have high 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 10 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 9 and 10, IEC constrains the UDP flow to its fair share (i.e., 49.9 Mbps).

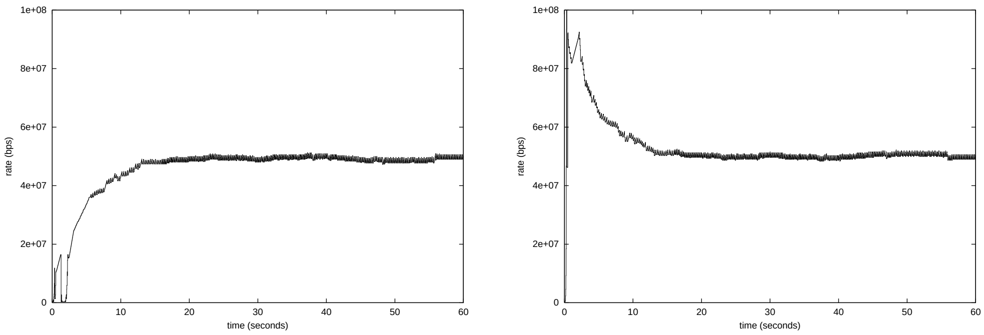
Figure 9 also shows that initially TCP lags because it's end-to-end feedback loop is larger than the edge-to-edge control loop. With a smaller control loop, the edge-to-edge virtual link adapts more quickly to demand than TCP adapts to available capacity. As a result, UDP initially provides all the demand and is given capacity that *TCP cannot immediately use.*

[1]Sending at four times the fair share.

Virtual[a] Links, N	Interior Support	Utilization	Queue Length[b] (avg/max pkts)	Drops[c] (pkts)	C.O.V. Throughput[d]
2	yes	98.8%	0.821/6	0	0.0236
2	no	99.0%	0.881/8	0	0.00445
10	yes	98.3%	8.58/31	0	0.0373
10	no	99.2%	9.30/34	0	0.0141
100	yes	96.4%	104.9/279	0	0.0448
100	no	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 in Variation of Throughput=$\frac{\text{standard deviation}}{\text{mean}}$

Figure 8: Single 100Mbps Bottleneck with UDP Sources



TCP Reno connection | UDP flow sending at 200Mbps

Figure 9: TCP versus UDP. Both regulated by Implicit Edge Control (IEC)

4.3 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 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.

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 [12].
[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 10: TCP versus UDP statistics

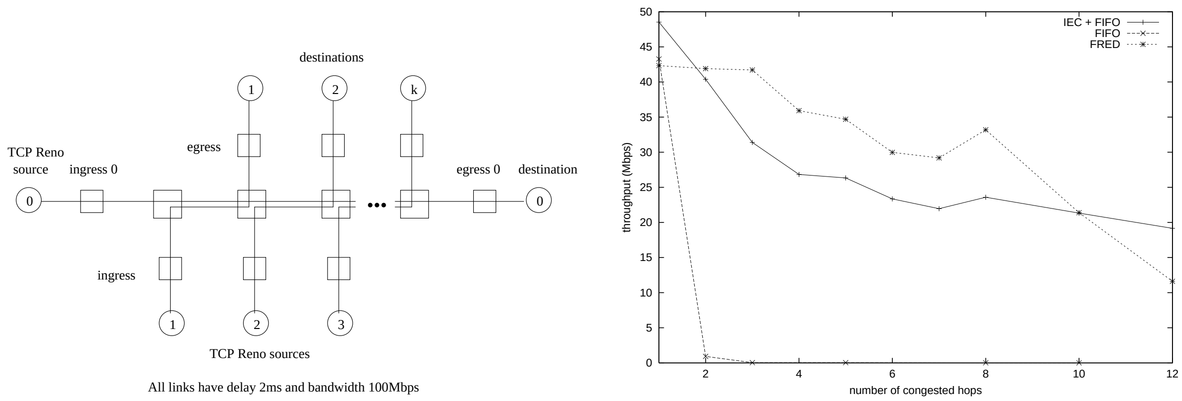


Figure 11: IEC experiences less beat-down than either FIFO or FRED.

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.

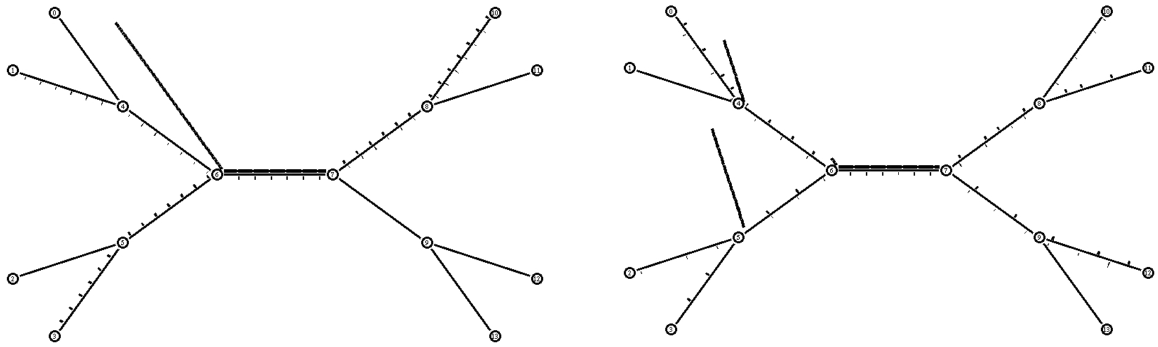
5.1 Distributed Buffer Management

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 [12]. As demonstrated in Figure 13(a), using IEC to distribute the backlog dramatically reduces the coefficient of variation in goodput[2] when many TCP connections

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

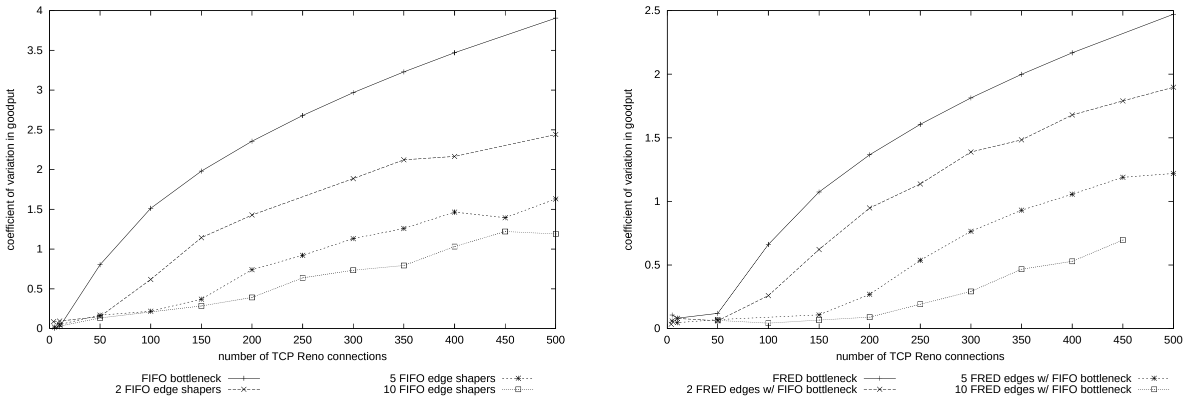
compete for the bottleneck. As expected this improvement increases as we distribute the congestion across more edges.

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.



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

Figure 12: Backlog moved from Bottleneck to Edge



(a) FIFO bottleneck versus IEC/FIFO edges (b) FRED bottleneck versus IEC/FRED edges

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

5.2 Low-loss Best-Effort Service

Edge-to-edge control can also be combined with Packeteer TCP rate control (TCPR)[10] to provide a low-loss end-to-end service for TCP connections. By "low-loss" we mean that *the scheme typically does not incur loss in the steady-state.* Furthermore as with IEC alone, the combined IEC+TCPR scheme does not require upgrading either end-systems or the interior network.

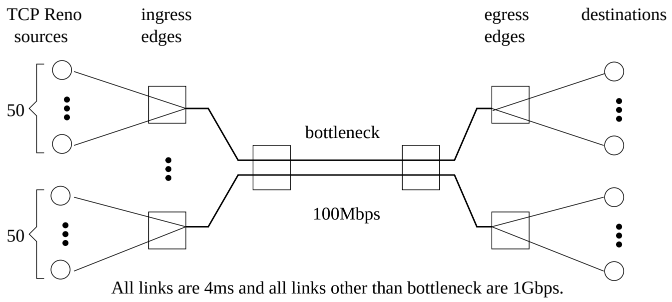


Figure 14: Topology used in low-loss and comparison simulations

In this combined scheme, IEC pushes the congestion to the edge and then TCP rate control pushes the congestion from the edge back to the source. To accomplish this the virtual link ascertains the available capacity at the bottleneck and provides this rate to the TCP rate controller. The TCP rate controller then converts the rate to the appropriate window size and stamps the window size in the receiver advertised window of acknowledgements heading back to the source.

The last row in Figure 15 demonstrates *IEC+TCPR operating without loss* over the topology in Figure 14. Furthermore it shows that TCP rate control can drastically reduce the backlog at the ingress edges when compared against IEC+FRED, and it can do this without loss of utilization. The reduction in backlog at the edges then leads to a reduction in end-to-end delay. Row 4 shows *IEC+TCPR reducing delay above propagation and transmission delays by 74%* compared to TCP-Reno over a pure FIFO network (i.e., row 1).

As a caveat this solution has a limited deployment space because the ingress edge must have access to the acknowledgement stream.

6 Deployment Issues and Future Work

As currently specified, both the Explicit Edge Control (EEC) and the Implicit Edge Control (IEC) schemes can be deployed independently in traffic control domains to improve the performance seen by flows passing through that domain (assuming the domain contains bottlenecks). Deployment can be piggybacked with the rollout of differentiated services or MPLS, since these techniques can work with either architecture. Both schemes are transparent to end-systems, but require software

Ingress Type	Bneck Type	Util-ization	Ingress Queue[a] (avg/max pkts)	Bneck Queue[b] (avg/max pkts)	Avg Delay[c]	Ingress Drops	Bneck Drops
FIFO	FIFO	99.5%	small/1[d]	202.5/375	36.4ms	0	4528
FIFO	FRED	100%	small/1[d]	181.2/206	34.5ms	0	7447
IEC+FRED	FIFO	99.1%	178.7/205	0.853/7	48.9ms	4771	0
IEC+TCPR	**FIFO**	**99.0%**	**19.52/53**	**1.015/8**	**24.2ms**	***0***	***0***

[a]Max and mean queue length across both ingress edges

[b]Bottleneck queue length

[c]Average end-to-end delay. Propagation delay is 20ms.

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

Figure 15: Comparing against low-loss service without interior support.

components to be installed at the edges of the network. Such edge components can be installed as upgrades to routers or stand-alone boxes. Also since we assume in our analysis that edge-to-edge controlled traffic is isolated from other traffic, it would be useful to provision a separate class in interior network queues for this traffic.

The EEC scheme requires upgrading key interior routers to enable promiscuous marking of packets during congestion epochs. Some implementations of RED[6] can be configured to act as a promiscuous marker, by setting $min_{th} = max_{th}$, 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.

The IEC scheme does not require support from interior nodes, but it requires some extra support from edge components. Though the IEC scheme does not require interior support, this approach needs more work to be extended end-to-end. This is because our scheme still assumes some domain knowledge such as maximum edge-to-edge RTTs etc. A straightforward method is to replace it with the measured RTT between the two edges running the control loop. If multiple independent, possibly overlapping edge-to-edge loops are present, we need further study to ensure that these loops don't have negative interactions between themselves or with other end-to-end flows which are not controlled thus. Currently, we find that since our definition of congestion differs from that of TCP, TCP is able to grab a larger share of bandwidth in competition with edge-to-edge controlled virtual links. This is because our scheme detects congestion early and continuously backs off even if there is no packet loss, whereas TCP waits for packet loss before backing off. We plan to address these issues in future papers and propose the scheme to the IETF ECM working group for consideration in end-to-end congestion management for a combination of TCP and non-TCP flows.

We have developed "edge discovery algorithms" to map end-to-end flows to edge-to-edge virtual links. We omit them here for lack of space, but specify them in our tech report [OMITTED].

References

[1] 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.

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

[3] 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.

[4] 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.

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

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

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

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

[9] Kalyanaraman, S., Jain, R., Fahmy, S., Goyal, R., and Vandalore, B. The ERICA switch algorithm for ABR traffic management in ATM networks. Submitted to IEEE/ACM Transactions on Networking, November 1997.

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

[11] 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.

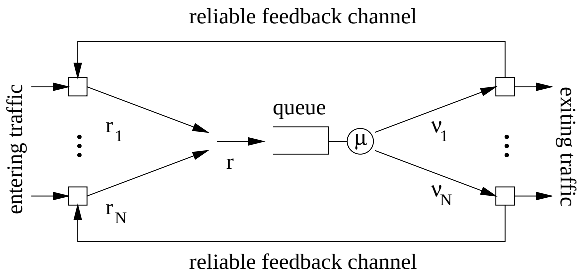
[12] Lin, D., and Morris, R. Dynamics of random early detection. In *Proceedings of SIGCOMM'97* (1997).

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

[14] 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.

[15] 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 Steady-State Queue Length Bound



$r(t)$	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 16: Topology, Constants, and Variables used in Analysis

We use a simplified fluid model to derive a hueristic bound on the bottleneck queue length for the Explicit Edge Control(EEC) scheme. In this model, we use the topology in Figure 16. 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
- $\mathsf{T} >$ 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 has a leaky bucket obeying inequality 1. With N flows this inequality becomes:

$$I[t, t+\tau] \leq \int_{t}^{t+\tau} r(T)dT + N\sigma \tag{2}$$

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:

$$r(t) \leq Na(t - t_\mu) + \mu \tag{3}$$

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

$$q(t+\tau) = q(t) + I[t, t+\tau] - \mu\tau \tag{4}$$

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 $t = 1$.

$$q(1) = N\sigma + I[0,1] - \mu \leq N\sigma + (\frac{Na}{2} + r(0) + N\sigma) - \mu \tag{5}$$

All terms appearing on the right-hand-side of inequality (5) 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 $[t_\mu, 0]$. Thus,

$$\int_{t_\mu}^{0} Na(T - t_\mu)dT = N\sigma, \quad t_\mu = -\sqrt{\frac{2\sigma}{a}} \tag{6}$$

$$r(0) = N\sqrt{2\sigma a} + \mu \tag{7}$$

Combining equation 7 and inequality 5 yields:

$$\begin{aligned} q(1) &\leq \frac{Na}{2} + N\sqrt{2a\sigma} + 2N\sigma = \left(\frac{5}{2} + \sqrt{2}\right)N\sigma && (8) \\ \text{max q} &\leq q(1) < 4N\sigma && (9) \end{aligned}$$

B Implicit Edge Control (IEC) Algorithm Flow Charts

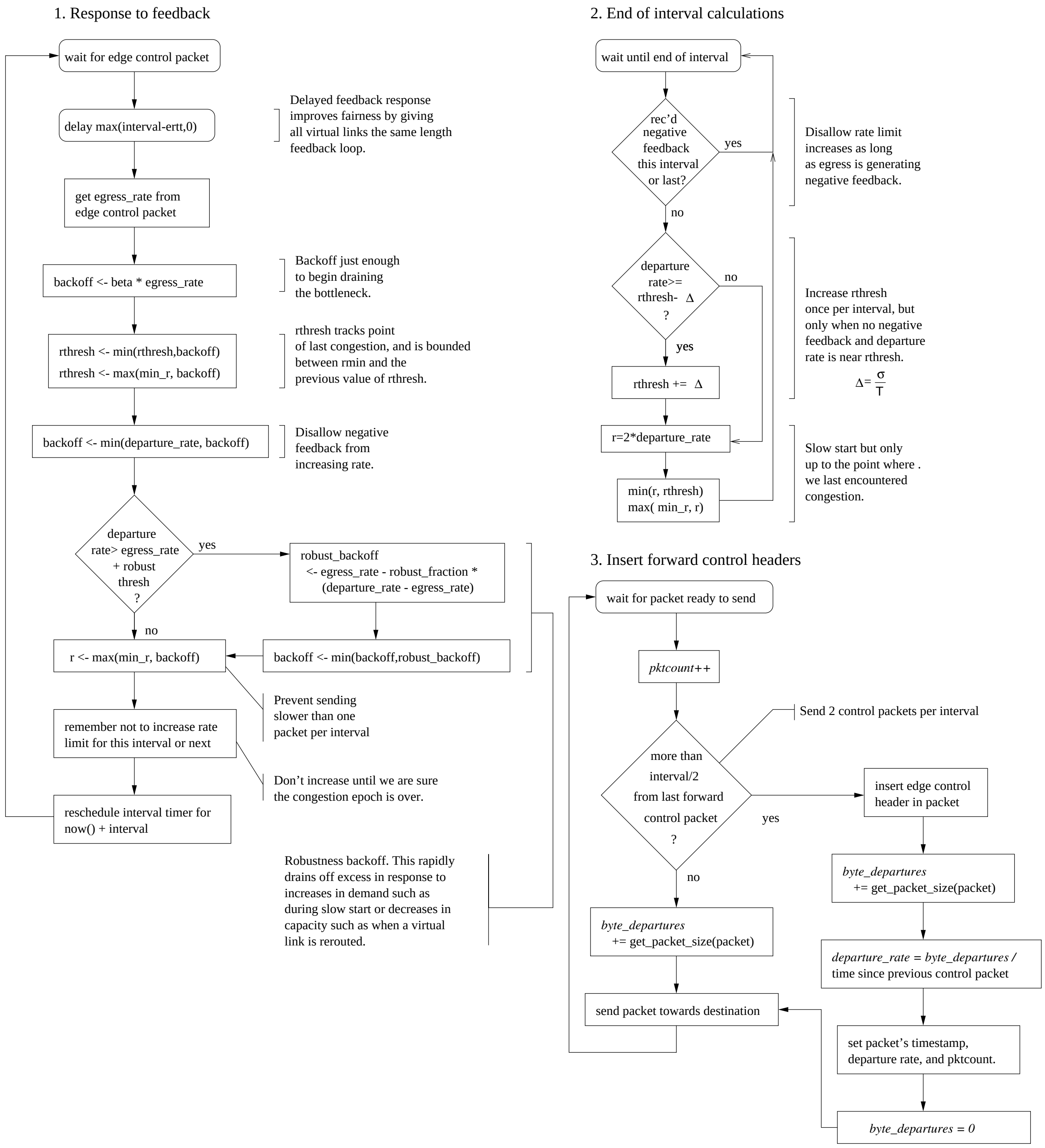


Figure 17: Ingress Control Flow Diagram for IEC

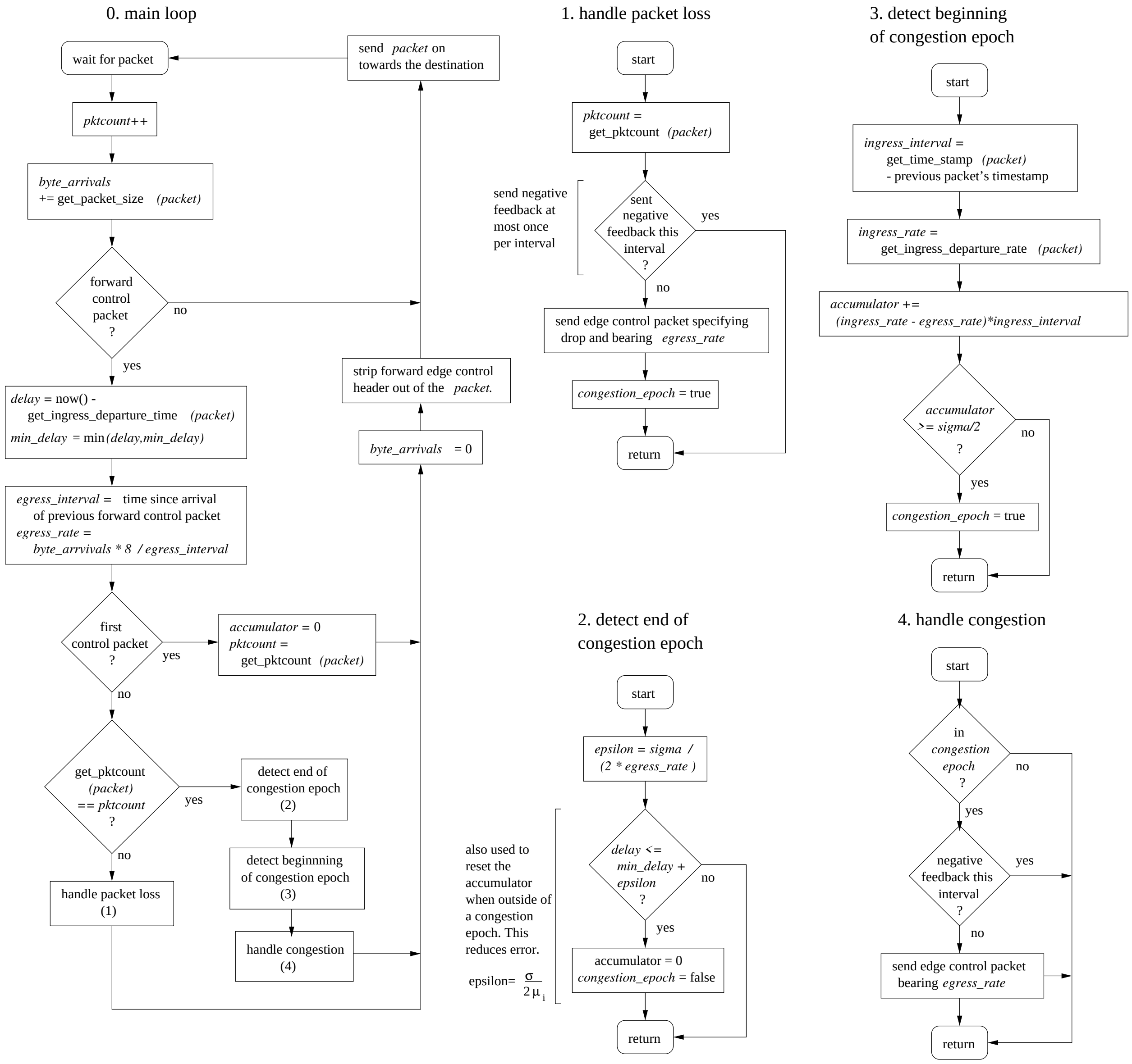


Figure 18: Egress Control Flow Diagram for IEC