Towards a Queue Sensitive Transport Protocol

Ritesh Kumar and Jasleen Kaur
University of North Carolina at Chapel Hill

Abstract

The TCP NewReno congestion control protocol relies only on packet losses for detecting congestion—this causes long NewReno transfers to build up large packet queues in router buffers. High buffer occupancy hinders performance of real-time applications as well as the development of high-speed routers. Several alternate congestion-control strategies have been proposed in the literature to help maintain low buffer occupancy. In this paper, we experimentally evaluate prominent proposals by emulating empirically-derived traffic mixes on a Linux-based lab testbed. We show that when existing proposals are used with such representative traffic mixes, they are unable to simultaneously ensure small router queues and high TCP throughput. Further analysis shows that the common practice of using TCP round-trip times for estimating queuing delays (as in TCP Vegas) fails in a highly-aggregated environment that contains short as well as long transfers—such environments would need to rely on explicit router feedback. We also find that proposals that rely on router feedback in the form of link utilization (as recently proposed in VCP) are not effective in maintaining high transfer throughput. Instead, we argue that a congestion control algorithm which uses queuing delay feedback from routers can reduce the buffer occupancy of routers without sacrificing response time performance of connections. We use this idea to design two new protocols—EDN and PEDN—and show that these two protocols can help manage the trade-off between maintaining low buffer occupancy and providing high TCP throughput.

1 Introduction

TCP is the dominant transport protocol used in the Internet. Given its wide-spread usage, the data transmission behavior of TCP congestion control algorithms fundamentally impacts Internet traffic dynamics. TCP senders widely employ the NewReno algorithm (or one of its several variants) [2, 9, 11, 18]. By design, NewReno uses a loss-based congestion-avoidance policy that tends to fill up router buffers at bottleneck links and suffers losses before it detects network congestion. This approach is undesirable for two reasons:

1. *Large buffers in high-speed routers are expensive.* A popular rule-of-thumb for provisioning router buffers is to set these equal to the bandwidth-delay product (BDP) of the network [23]. The basic goal is to maintain high link utilization with TCP. However, the BDP for high-speed Internet routers can be quite large and provisioning routers with such large buffers is expensive both in terms of design and manufacturing [3].
2. *Large queuing delays hinder real-time and interactive applications.* The performance of end-to-end protocols for interactive and real-time applications such as voice-over-IP [22] and network games is quite sensitive to large network delays [4, 25]. Large router queues result in large queuing delays and do not support such applications well.

Several backbone ISPs take the approach of over provisioning their link capacities to avoid queue buildups [19]—many have been reported to run their networks at less than 50% utilization. However, given the rapid increase in application data rates as well as the capacities of edge networks, it is not clear if ISPs can continue to do so in the future.[1] Moreover, networks of regional and local ISPs are seldom significantly over-provisioned. Given the rapid rise in capacities of edge networks and application data rates, we believe it is desirable to design a network architecture in which ISPs are able to run their networks at high levels of utilization without adversely impacting customer performance. This can be realized with the help of queue-friendly congestion-control protocols.

Several queue-friendly designs for congestion-control have been proposed in the literature—including end-to-end delay-based congestion avoidance protocols (e.g., Vegas [6]) as well as protocols that rely on explicit router feedback (eg. AQM+ECN and VCP [24]). In this work, we first experimentally evaluate prominent congestion control proposals to see how well they support both low queuing activity in routers and high throughput in TCP transfers. For

[1] In fact, one major national provider has acknowledged operating their current links with even up to 90% utilization. This information has been obtained through a personal communication with the provider. Non-disclosure agreements prevent us from revealing its identity.

this, we rely on a lab testbed that emulates an empirically-derived traffic mix. Specifically, we use the Tmix traffic generator that faithfully reproduces in a testbed setting, the path round-trip times, connection-arrival processes, transfer sizes, as well as the application-level socket writing behavior, as observed in traffic carried by production Internet links. We run the transfers emulated by Tmix over several different congestion-control protocols and observe the behavior of router queues as well as per-transfer throughput.

We find that existing designs either do not succeed in maintaining small router queues, or do so at the expense of throughput of individual transfers. Our analysis leads to several important guidelines—including the need for explicit router feedback, as well as the appropriateness of queuing delay as a feedback metric—for ensuring a design that is simultaneously queue-friendly as well as throughput-sustaining. Based on these guidelines, we design two new congestion-avoidance protocols, referred to as Explicit Delay Notification (EDN) and Per-flow EDN (PEDN), both of which help provision small router buffers by successfully maintaining fairly small bottleneck queues. The two designs let the adopter choose between the ability to maintain small router queues independent of the load or degree of traffic aggregation, versus the ability to ensure NewReno-like throughput for individual transfers but with queue buildups that grow moderately at very high loads.

The rest of this paper is organized as follows: we describe related work in Section 2 and our methodology in Section 3. We present evaluation results for existing protocols in Section 4. We motivate, describe, and present evaluation results for EDN/PEDN in Section 5. Finally, we conclude in Section 6.

2 Related Work

Router buffer provisioning Network operators and router manufacturers must address the issue of deciding how many packet buffers to provision in the routers. A popular rule-of-thumb recommends using the *bandwidth-delay product* (BDP) to calculate the size of router buffers [23], where the bandwidth is the transmission capacity of the outgoing router link, and delay is the average round-trip propagation delay experienced by connections going through the router. The intuition is that such buffers are needed to maintain a high link utilization in the presence of the saw-tooth behavior of TCP congestion-control [23].

Recently, there has been considerable interest in the performance of loss detection based congestion control (mainly NewReno) with small router buffers [3, 12]. These studies argue that as the number of long NewReno transfers aggregated on a router link increases, the amount of buffers needed to ensure high link utilization decreases. There are two main problems with this approach: (i) it is difficult to estimate the number of "long lived" connections going through a router on average (for calculating the size of buffers needed) [12], and (ii) the NewReno transfers suffer high loss rates (and a corresponding degradation in response times) if the load on the router is more bursty than what the provisioned buffers can handle [8, 7, 20].

In this work, we tackle bursty network dynamics by adopting congestion control algorithms that reduce router buffer *occupancy*, which in turn helps reduce demands on router buffer *provisioning*. In fact, we propose two variants that represent the two ends of a trade-off; EDN: which moderately sacrifices response times at high loads to maintain consistently low queue build-ups, and Per-flow EDN (PEDN): which allows moderately higher queuing activity but also maintains NewReno-like connection response times, even at fairly high loads. We revisit this trade-off in Section 5.1.

Delay based end-to-end congestion control The idea of using high delays, rather than only losses, as indicators of network congestion has been explored in the past—the most popular proposal is that of TCP Vegas [6]. Since such protocols do not need to fill up buffers (and experience losses) before they reduce their sending rates, these are good candidates for maintaining low buffer occupancy. However, our experimental evaluation in Section 4 shows that Vegas does not significantly reduce queue buildups in an aggregated network environment. We subsequently show that this is because of the inability of Vegas to accurately detect trends in router queuing delays from just the end-to-end RTTs. In EDN/PEDN, routers explicitly send the queuing delays as feedback to TCP senders—this helps maintain small router queues.

Router feedback based congestion control The use of explicit router feedback for guiding TCP congestion control has been proposed by several others [21, 17, 24, 16]. ECN [21, 17] is the most extensively deployed mechanism that can be used by routers for sending congestion-related feedback to TCP senders. ECN is intended to be used in conjunction with Active Queue Management (AQM) schemes [10, 5, 14] that predict the onset of congestion by monitoring router queue lengths, and use ECN to notify senders of early congestion.

XCP and VCP are recently-proposed schemes that use more sophisticated feedback than simply an ECN bit—these have been shown to perform better than AQM+ECN schemes in reducing packet loss rates and average queue buildups [16, 24]. XCP [16] targets high-speed networks; routers in an XCP network indulge directly in TCP congestion control and use link utilization, queue sizes, and a computation of per-connection fair share of bandwidth, to inform the sender of how to update the congestion window. VCP [24] is a simpler form of XCP and has been shown to perform as well. It uses link utilization as an indicator of congestion and proposes making ECN a two-bit

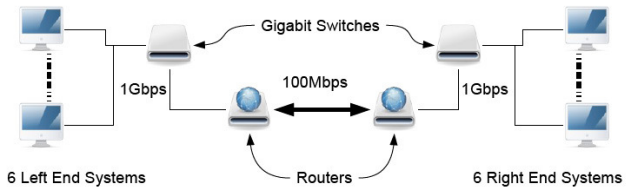


Figure 1. Experimental Testbed

notification. It also modifies the TCP congestion control to increase congestion-windows more aggressively—a VCP sender does not exit the multiplicative Slow Start phase until the router explicitly indicates that the utilization is more than 80%; it does not exit additive increase until the router is 100% utilized or a packet is lost.

In Section 4.2 we show that, despite being an explicitly guided protocol, VCP fails to maintain small queue sizes with aggregate traffic mixes—our analysis traces the cause as its aggressive congestion control algorithm. In all fairness, VCP is designed for high-speed networks; maintaining small router queues is not a primary design goal. We then remove the aggressiveness from VCP congestion control and show that when routers guide non-aggressive TCP congestion control using link utilization as the feedback metric, they are unable to maintain high link utilization. Based on these observations, we propose to use queuing delay as the feedback metric with a non-aggressive congestion control algorithm—our experimental evaluations show that this helps maintain small router queues without limiting link utilization or per-transfer throughput.

3 Experimental Methodology

In this paper, our main objective is to study the interaction between TCP congestion control and router buffer occupancy. Several traffic characteristics like offered load, degree of aggregation (number of transfers sharing a router link), round-trip times (RTTs), and application data transfer patterns are likely to impact the nature of this interaction as well. Our experimental evaluation methodology in this paper is guided by the goal of ensuring that these characteristics are incorporated in a manner that is *representative* of how they occur in the Internet. For this we rely on the Tmix traffic generator [13] that empirically derives distributions of several of these characteristics from traces collected on production Internet links, and reproduces them in a controlled lab setting.

We generate thousands of TCP connections using Tmix and evaluate their performance in a controlled lab setting (as against relying on simulations). Below, we describe our testbed, instrumentation, and Tmix.

Testbed Figure 1 illustrates our experimental testbed that sets up a dumbbell topology using two routers (IBM dual Xeon 1.8Ghz, 1.2GB RAM), 2 1-Gbps switches, and 12 end-hosts (IBM Pentium III 800Mhz - 1Ghz, 1GB RAM). All links are 1Gbps, other than the 100Mbps link connecting the two routers—this is the bottleneck link of our topology. All nodes and routers run Linux 2.6.20 (released Feb 4th, 2007). Unless otherwise stated, the router is configured with a 1.3MB software byte-FIFO queue. This corresponds to the delay-bandwidth product of the bottleneck link (the median RTTs of the emulated TCP connections is around 100ms).

Instrumentation To study queuing activity we tap into the software byte-FIFO queue implementation in Linux. We time-stamp *every* packet arrival and departure in a "queue log". To time-stamp packets accurately, we rely on the processor time-stamp counter (TSC) [15] which is an accurate low overhead clock. The device driver's packet queue size was reduced to 2 so that we could see almost all the queuing in the IP layer's byte-FIFO queue.

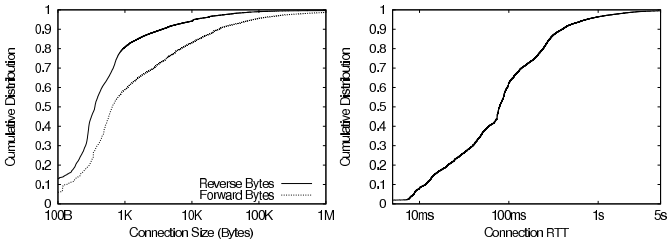
We also modified Linux to report back the average RTT (from the same samples which TCP uses for its RTO estimation) on a per connection basis using *getsockopt()*.

Tmix Traffic Generator Tmix [13] is a recently-developed application-level traffic generator that helps us control the TCP congestion control algorithm and traffic load while keeping the rest of the traffic characteristics invariant and representative across experiments. Tmix can read in a trace of packet headers collected from real Internet links and derive from it: (i) the exact sequence in which data units are transmitted in either direction of each TCP connection (including the "pauses" or user think times during which no data flows in either direction), (ii) the per-connection minimum RTT, (iii) the per-connection limits on socket buffer sizes, and (iv) the exact start times and sizes of TCP connections. The first feature implies that Tmix is able to infer for each TCP connection, the socket-level data transmission activity of the corresponding connection. This connection level meta-data is referred to as a "tcvec".

Once these per-connection characteristics are derived, Tmix can be configured to generate traffic on a given testbed according to a tcvec-set. For each tcvec (connection) in a tcvec-set, Tmix then generates a real TCP connection between a pair of the testbed machines, that accurately reproduces each of the above characteristics. Since, Tmix operates above TCP sockets, it gives us complete freedom to modify the underlying TCP protocol and router mechanisms. To reproduce per-connection RTTs, Tmix uses a kernel-level module that artificially delays packets using a specified per-connection delay.

Tmix lets us control the "offered load" for the generated traffic. The offered load refers to the average traffic rate generated by a tcvec set on an uncongested network.[2]

[2]For our testbed, this would be the traffic load on the network if the bottleneck link had a capacity of 1Gbps or higher.



(a) Distribution of Transfer Sizes (b) Distribution of Transfer RTTs

Figure 2. Trace properties

Tmix derives from a tcvec-set a new tcvec-set with a different offered load by randomly sampling connections from the original tcvec-set—it does so while preserving the short-term connection-arrival patterns of co-existing connections. Tmix has been exhaustively validated in [13].

Tcvec Set To generate tcvec-sets, we used 3 different types of packet traces: (i) one collected on a 1Gbps link connecting a major US university campus to the Internet, (ii) a web trace collected on a 1Gbps access link of a cluster of high-traffic web-servers (ibiblio.org), and (iii) an ISP trace collected from a 2.5Gbps peering link of a major US ISP. Our observations are qualitatively similar for each of the above traces, even though the traces themselves exhibit fairly different connection size distributions and number of simultaneous connections—for brevity, we include results only for the first trace in this paper.[3]

Figure 2 plots the distributions of connection sizes and per-connection RTTs observed for connections in this trace—as can be expected from Internet traffic, the connections in these traces are fairly diverse. The offered load of the generated tcvec was around 80Mbps. We derived 3 additional tcvec-sets from this at offered loads of 85Mbps, 90Mbps, and 95Mbps each. The reverse-path load is small for each of these (around 19Mbps).

At the end of an experiment, for each connection we compute the performance metric of "data response time"—this is the total time that is spent in transferring data in the connection. It is computed as the total duration of the connection minus the total user think time.

4 Evaluation of Existing Protocols

We begin be studying the queuing activity induced by the NewReno, Vegas and VCP protocols, when used to carry the Tmix TCP connections. Figure 3 plots the distribution of the router queue size (sampled on every enqueue event) for all the congestion control protocols at 90Mbps offered load. We first observe that queues can grow quite large (more than a mega-byte) with NewReno. The large queuing activity is surprising given that our tcvec-set consists

[3]These tcvec-sets are also available for download at http://www.cs.unc.edu/~jasleen/research/edn/.

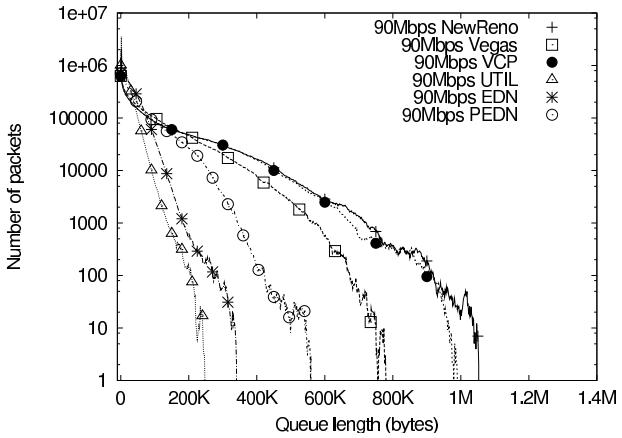


Figure 3. Queue distribution

mostly of small connections (Figure 2). This indicates that even when only a small fraction of connections are long-lived, they are capable of sustaining a high router buffer occupancy.

We also find that queuing activity is high even with Vegas and VCP, despite their queue-friendly designs—we explore the reasons for this in the rest of this section. Figure 5, which plots the distribution of per-transfer response times, shows that there is negligible difference in the connection response times observed with NewReno, Vegas, and VCP.

4.1 Vegas

To understand why Vegas does not reduce queuing activity, despite using delay based congestion avoidance, let us take a closer look at its congestion control algorithm. Vegas uses RTT samples to infer the router queuing delay. A Vegas connection estimates the current bottleneck queuing delay as: $currRTT - baseRTT$, where $currRTT$ is the latest value of RTT estimated by the connection, and $baseRTT$ is the minimum RTT that the connection has sampled so far (which it assumes to be close to the round-trip propagation delay for the connection's path). The problem with Vegas is that $baseRTT$ may be a fairly crude approximation for the round-trip propagation delay of a connection.

To illustrate this, we plot in figure 4 the CDF of $baseRTT/propagationRTT$ for all connections emulated in the 90Mbps and 95Mbps Vegas experiments (where $propagationRTT$ is the round-trip propagation delay for a given connection, as emulated by Tmix on the testbed). We see that this ratio can be quite large for a significant fraction of connections—this shows that $baseRTT$ can be much larger than the actual round-trip propagation delay. This can happen if a router in the path experiences *persistent queuing* in which case the Vegas connection never samples an empty queue and hence is not able to get a good estimate for $baseRTT$.

When $baseRTT$ is much larger than $propagationRTT$, Vegas ends up underestimating its contribution to the queu-

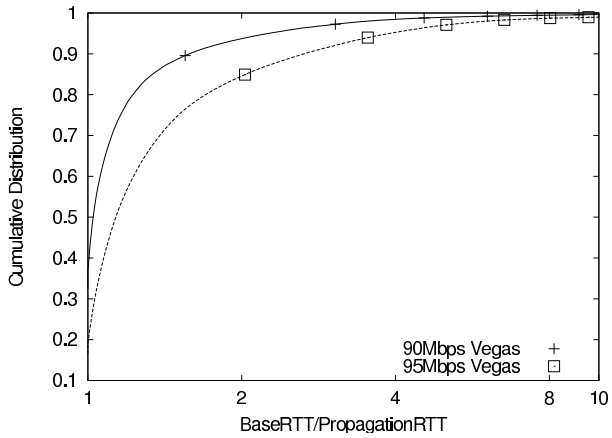


Figure 4. baseRTT vs. propagationRTT

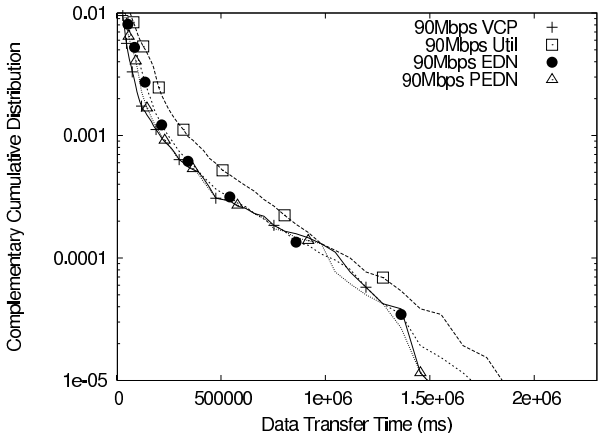


Figure 5. Response time CCDF with UTIL

ing activity and sustains large queue buildups. This suggests that *congestion control protocols should rely on explicit router feedback (as against operating on a purely end-to-end basis) in order to detect the presence of large queues.*

4.2 VCP

Figure 3 shows that even VCP, despite the use of explicit router feedback, can cause a large buildup in the router queue. However, it is important to note that other than the choice of explicit versus implicit feedback, VCP differs from Vegas in two significant aspects: the performance metric used for detecting congestion (utilization vs. heuristically inferred queuing delay), and the congestion-control algorithm. VCP differs significantly from Vegas in the latter aspect—it targets high-speed networks and is, therefore, quite aggressive in increasing its congestion window. A VCP sender multiplicatively increases its send rate when the link utilization based feedback is smaller than 80%. It uses additive increase if the feedback is between 80% and 100% and does a multiplicative decrease only when the feedback is greater than 100%. This aggressive behavior may contribute to queue buildups.

In order to evaluate if the aggressive congestion control of VCP is indeed responsible for its large queuing activity, we isolate and study the choice of router feedback metric alone. For this, we develop and implement a protocol that uses the VCP feedback metric based on link utilization, but employs a Vegas-like congestion-control response—we refer to this congestion control algorithm as UTIL. Specifically, the UTIL sender updates its congestion window (*Cwin*) as follows:

if feedback > 0.8,	exit Slow Start;
if feedback < 0.8,	Cwin++;
else if feedback ≥ 1.0,	Cwin−−;

In addition to the above logic, UTIL (like VCP) retains the semantics of NewReno on experiencing a loss and multiplicatively reduces Cwin.

Experiment	Link Util.	Experiment	Link Util.
90Mbps Reno	90.75%	90Mbps EDN	90.85%
90Mbps Vegas	90.74%	90Mbps PEDN	90.73%
90Mbps VCP	90.71%	90Mbps UTIL	89.51%

Table 1. Link Utilization for Experiments

Figure 3 plots the queue occupancy observed with UTIL under 90Mbps offered load. We find that UTIL successfully maintains a fairly low queue buildup—in fact, the occupancy is lower than that observed with any other congestion control algorithm. This suggests that indeed, VCP's MIMD-based congestion control algorithm was responsible for its queuing activity. In order to be queue-friendly, therefore, *a congestion control algorithm should adopt a non-aggressive window-increase policy*.

Unfortunately, as argued before in Section 2, the use of utilization as a metric is likely to reduce the utilization achieved when the offered load is high. Table 1, that lists the utilization achieved in each experiment, confirms this. The reduction in link utilization also impacts the connection response times. Figure 5 shows that UTIL can increase the response times of connections by 20-30% (read as horizontal gap between the CCDFs), especially for long lived connections. Thus, we conclude that in order to maintain high link utilization and low connection response times, *an explicitly guided congestion control protocol should rely on queuing delay instead of link utilization as the feedback.*

5 EDN

Our evaluations in Section 4 lead to the following insights:

- RTTs do not yield accurate estimates of router queuing delays. A congestion-control protocol that aims at reducing queue buildups should rely on *explicit router*

feedback for this.

- An aggressive congestion-control policy tends to cause large queue buildups—protocols should rely on a non-aggressive policy, such as additive-increase/additive-decrease (AIAD).
- Using a link utilization based feedback metric with an AIAD congestion avoidance algorithm leads to degradation in connection throughput performance and link utilization—the feedback should be a direct measure of queuing activity.

We use the above insights to design a new congestion control protocol—referred to as Explicit Delay Notification (EDN)—that helps reduce queuing activity at routers. EDN uses an *AIAD congestion avoidance* algorithm with *explicit router feedback based on queuing delays*.

Explicit Router Feedback in EDN Routers in an EDN network encode two values in the IP packet header for conveying queuing delays back to TCP senders. These are D_f, that encodes the total queuing delay on the forward path of a packet, and D_r, that encodes the total queuing delay on the reverse path. A TCP sender initializes the D_f field of each new packet to 0. For each incoming packet, a router observes the current queue length and uses its link capacity to compute the queuing delay that would be seen by the packet. It then adds it to the packet's D_f before enqueuing it. This process is repeated at all routers in the path of a packet.

When the TCP receiver receives the packet, it copies the D_f field into the D_r field of the next ACK packet it sends to the sender. The D_f field of this new packet is initialized to 0 and set appropriately by routers on the path to the sender. When the TCP sender receives this ACK, it obtains the forward path queuing delay from the D_r field and the reverse path queuing delay from the D_f field. The sender then adds them up to compute the *round-trip queuing delay* on the path. The sender also copies the D_f field into the D_r field of the next packet it transmits.

A big advantage of the above feedback scheme is the simplicity of its implementation and its low run-time execution cost, both of which are important considerations in high speed routers.

Delay-based Congestion Control in EDN EDN's congestion control algorithm uses AIAD congestion avoidance like Vegas. However, it uses only two parameters A and B (instead of Vegas' three parameters α, β and γ). Also, instead of sampling and using the $curRTT$ feedback only once every RTT, with EDN we sample and use the queuing delay feedback on every packet[4]. Our EDN congestion avoidance algorithm is a simple modification to New Reno's congestion control algorithm as shown below (EDN additions are in boldface):

```
if(Cwin ≤ ssthresh)
  if(round-trip queuing delay > B) exit slow start
  else do slow start
else
  if(round-trip queuing delay < A)
    Cwin_count++
  else
    Cwin_count- -
  if(Cwin_count ≥ Cwin) Cwin++ ; Cwin_count = 0
  if(Cwin_count ≤ -Cwin) Cwin- - ; Cwin_count = 0
```

where A and B (A < B) are parameters which we set to be 2ms and 3ms. In addition to the above congestion avoidance algorithm, EDN retains the semantics of NewReno on experiencing a loss and multiplicatively reduces Cwin.

One of the differences between EDN and most of prior work involving router feedback is that an EDN-enabled router does not apply any low-pass filters to the feedback before sending it to end-systems. We believe that the congestion control algorithm running on the end-systems is the best judge of how to interpret the router feedback signal based on its RTT and congestion-control mechanisms. An EDN sender applies a "smoothing" function to the feedback by accumulating jitter in the quantity: $Cwin\_count$.

Also note that EDN retains the RTT-clocked window-updating mechanism of all existing protocols. Thus, it adopts the fairness properties exhibited by window-based protocols—long transfers with shorter RTTs attain a higher throughput than long transfers with long RTTs. Achieving (RTT) fairness is not the explicit goal of this paper.

Prototype Implementation We have developed a prototype implementation of EDN in Linux [1] by modifying its NewReno implementation. For encoding router feedback, we rely on 14 bits available in the *fragmentation offset* of the IP packet header—since our testbed has the same MTU on all links, these fields are unused in all packets. We store D_f in the lower 7 bits and D_r in the higher 7 bits. Note that since the congestion control algorithm relies only on an inequality operator (with parameters A and B), we have been able to manage with only a few bits.

5.1 EDN Performance

Figure 3 shows that EDN is indeed capable of reducing queue buildups. Also, it does so with little to no difference in connection throughput performance or link utilization. This is illustrated by Figure 5, which shows that EDN, unlike UTIL, does not adversely impact the connection response times, even though it reduces queuing activity. Table 1 shows that EDN also maintains high link utilization.

[4]VCP/UTIL use per RTT feedback sampling like Vegas.

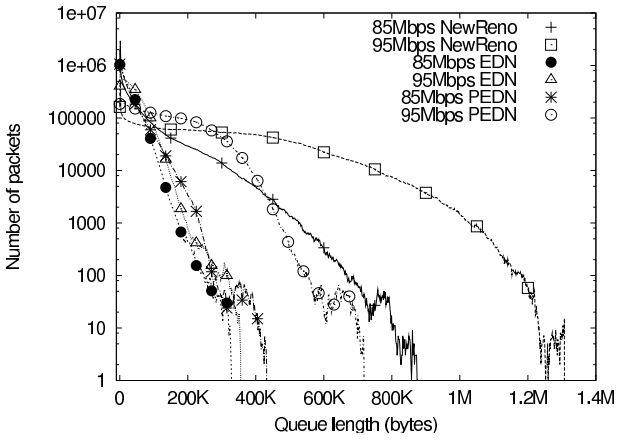


Figure 6. Queue distribution with EDN/PEDN

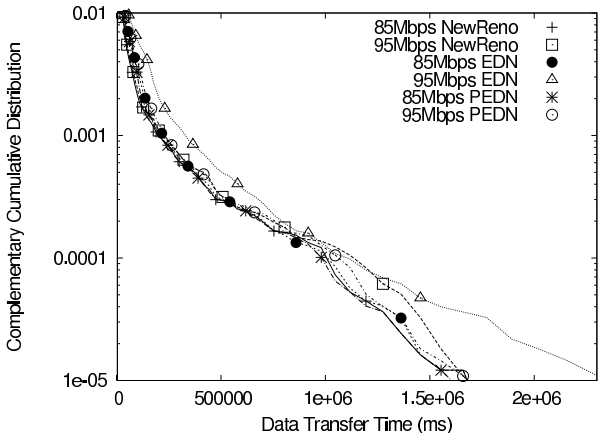


Figure 7. Response times (EDN/PEDN)

Figure 6 shows the queue distributions of EDN experiments for loads of 85Mbps and 95Mbps and compares it with queue distributions for NewReno. The queue distribution with EDN stays remarkably stable even with a significant increase in traffic load. This is to be expected as an EDN network is designed to explicitly maintain small queues irrespective of load.

Figure 7 shows the connection response times for the same experiments. The response times for EDN are close to those of NewReno for reasonably high loads of up to 85Mbps. However, at even higher loads of 95Mbps, the reduction in queues with EDN comes at the cost of a moderate degradation in the response times of long transfers. At such high loads, the number of simultaneously-active transfers is very high; consequently the per-transfer share of the router buffers is fairly small—this limits the ability of long transfers to sustain a reasonably high congestion window. Thus, EDN is designed to favor low queuing in the well-known trade-off between queuing delays and TCP throughput.

It is possible to imagine the other end of this trade-off, where we *allow moderately higher queuing activity to get NewReno like response times even for long transfers*. Such a protocol is certainly desirable from an end user perspective and only moderately increases costs due to queuing activity. To realize the above trade-off, we use as feedback the queue-contribution on a *per connection* basis instead of focusing on the queuing activity caused by the *connection aggregate*. Thus, we maintain a low value of the *per flow* queue contribution low; the total queue size at the router may then increase as the number of connections aggregated increases. This lets long transfers attain good throughput even under high degrees of aggregation. This leads us to our EDN variant called Per-flow EDN, which uses per-flow the queuing delay feedback to drive the EDN congestion control algorithm.

5.2 Per-flow EDN

The only major difference between PEDN and EDN is the use of *per-flow* queuing delay feedback by the former. With PEDN, only the packets of the same flow are used for calculating the queuing delay feedback for that flow. For fast lookup and update, we use a hash-table to store the *per-flow* number of packets in the queue using an IP-address/port 4-tuple based key. This does not guarantee complete isolation between flows but we found that this scheme works very well in practise. We use the per-flow number of packets directly for feedback instead of their queuing delay.

Note that the number of connections for which packets are likely to be in the queue at any point in time are much smaller than the total number of simultaneous connections going through a router. We hence size our hash-table proportional to the queue size instead of the number of simultaneous connections going through our bottleneck link. This helps us reduce the memory footprint of the hash-table significantly.

We retain EDN's congestion control algorithm but use it with parameters A and B set to (the transmission times of) 2 and 3 packets, respectively.

5.3 Per-flow EDN Performance

Figure 3 shows that PEDN is better than Vegas at reducing queuing activity. Very importantly, as shown in figure 5, there is little to no difference between the connection response times seen with PEDN and NewReno/Vegas/VCP.

With different offered loads (85Mbps and 95Mbps), PEDN again shows response times very close to NewReno but with significantly lower queuing activity. PEDN causes higher queue buildups than EDN, but succeeds in maintaining the performance of individual (especially long lived) connections (Figures 6 and 7). It is important to note that

in these experiments (that see virtually no packet loss), PEDN can not outperform the response-time performance of NewReno—this is because it exits slow start earlier and decreases its congestion window during congestion avoidance (unlike NewReno). PEDN is able to achieve NewReno-like response times because besides reducing $Cwin$, it also reduces the RTT of the connections (due to lower queuing delays). By maintaining a few packets in the queue for every connection, PEDN helps to keep congestion windows high enough for achieving a good response-time performance.

6 Conclusions

In this paper, we experimentally show that NewReno, Vegas and VCP congestion control protocols cause large queue buildups at routers that carry highly-aggregated and representative traffic mixes. We also show why these protocols are not able to achieve low queuing activity despite having queue-friendly design choices.

We propose two new congestion control algorithms EDN and PEDN and show how EDN and PEDN are at two ends of a trade-off: EDN moderately sacrifices connection response time performance to keep the queuing activity stable with increase in traffic loads; PEDN incurs moderately higher queuing activity with increase in traffic loads but maintains per-connection performance similar to NewReno.

PEDN shows that it is indeed possible to achieve lower queuing activity than NewReno/Vegas/VCP while maintaining the same connection throughput performance for both short-lived and long-lived connections with a representative mix of TCP traffic.

References

[1] http://www.kernel.org.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM*, Portland, Oregon, August 2004.

[4] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *IEEE International Conference on Networks (ICON)*, Sydney, Australia, September 2003.

[5] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. Rem: Active queue management. *IEEE Network*, 15(3):48–53, 2001.

[6] L. S. Brakmo and L. L. Peterson. Tcp vegas: end to end congestion avoidance on a global internet. *JSAC - IEEE Journal on Selected Areas in Communications*, 13(8):1465 – 1480, October 1995.

[7] A. Dhamdhere and C. Dovrolis. Open issues in router buffer sizing. *CCR - ACM SIGCOMM Computer Communications Review*, 36, Issue 1:87 – 92, January 2006.

[8] A. Dhamdhere, H. Jiang, and C. Dovrolis. Buffer sizing for congested internet links. *INFOCOM - Conference of the IEEE Computer and Communications Societies*, 2:1072 – 1083, March 2005.

[9] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782 (Proposed Standard), Apr. 2004.

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

[11] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883 (Proposed Standard), July 2000.

[12] Y. Ganjali and N. McKeown. Update on buffer sizing in internet routers. *CCR: Computer Communications Review*, October 2006.

[13] F. Hernandez-Campos, F. D. Smith, and K. Jeffay. Generating realisic tcp workloads. In *CMG - The Computer Measurement Group*, pages 273 – 284, Las Vegas, Nevada, December 2004.

[14] C. V. Hollot, V. Mishra, D. Towsley, and W.-B. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proceedings of IEEE Infocom*, pages 1726–1734, Piscataway, NJ, USA, 2001.

[15] Intel. *Intel 64 and IA-32 Architectures: Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z*, November 2006. RDTSC: p246–247.

[16] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM SIGCOMM*, Pittsburgh, Pennsylvania, August 2002.

[17] A. Kuzmanovic. The power of explicit congestion notification. In *Proceedings of ACM SIGCOMM*, Philadelphia, Pennsylvania, August 2005.

[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), Oct. 1996.

[19] K. Papagiannaki, S. B. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot. Analysis of measured single-hop delay from an operational backbone network. In *Proceedings of IEEE INFOCOM*, Piscataway, NJ, USA, 2002.

[20] R. Prasad, C. Dovrolis, and M. Thottan. Router buffer sizing revisited: The role of the output/input capacity ratio. In *Proceedings of the CoNEXT Conference*, December 2007.

[21] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), Sept. 2001.

[22] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320.

[23] C. Villamizar and C. Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, 1994.

[24] Y. Xia, L. Subramaniam, I. Stoica, and S. Kalyanaraman. One more bit is enough. In *Proceedings of ACM SIGCOMM*, Philadelphia, Pennsylvania, United States, August 2005.

[25] S. Zander, I. Leeder, and G. Armitage. Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 117 – 124, Valencia, Spain, 2005.