Multimedia-unfriendly TCP Congestion Control and Home Gateway Queue Management

Lawrence Stewart, David A. Hayes, Grenville Armitage
Centre for Advanced Internet Architectures,
Swinburne University of Technology
Melbourne, Australia
{lastewart, dahayes, garmitage}@swin.edu.au

Michael Welzl, Andreas Petlund
Department of Informatics,
P.O. Box 1212, University of Oslo
University of Oslo, Norway
{michawe, apetlund}@ifi.uio.no

ABSTRACT

Consumer broadband services are increasingly a mix of TCP-based and UDP-based applications, often with quite distinct requirements for interactivity and network performance. Consumers can experience degraded service when application traffic collides at a congestion point between home LANs, service provider edge networks and fractional-Mbit/sec 'broadband' links. We illustrate two key issues that arise from the impact of TCP-based data transfers on real-time traffic (such as VoIP or online games) sharing a broadband link. First, well-intentioned modifications to traditional TCP congestion control can noticeably increase the latencies experienced by VoIP or online games. Second, superficially-similar packet dropping rules in broadband gateways can induce distinctly different packet loss rates in VoIP and online game traffic. Our observations provide cautionary guidance to researchers who model such traffic mixes, and to vendors implementing equipment at either end of consumer links.

1. INTRODUCTION

Most home Internet users connect to their Internet Service Provider (ISP) through asymmetric consumer broadband links. These links experience a mixture of traffic generated by latency-tolerant applications (such as web browsing, streaming multimedia, and peer-to-peer content transfer) and latency-sensitive applications (such as voice over IP (VoIP) and online interactive games). For latency-sensitive applications, transmissions are usually triggered by interactions between users (or users and a system). The result of such interaction patterns is that many streams produced by interactive applications show very small packet sizes and (relatively) high interarrival times between packets. Table 1 shows examples of such thin-stream applications and how their packet size and interarrival times compare.

Greedy traffic sources tend to probe and utilise as much path capacity as they can get away with. Consequently their behaviour "on the wire" can involve wide variations in inter-arrival times, even down to sequences of back to back packets, and packet sizes are often distributed bi-modally between pathMTU and smallest IP packet size.

Loss of packets will in many cases reduce the experience of the interactive application e.g. introduce noise into a VoIP conversation [1]. In other applications, like online games, congestion-induced latency can delay response time for the game enough to significantly reduce game performance[1].

Many consumer modems provide minimal configuration controls for upstream quality of service (QoS) at the home end, and non-existent end-user control of QoS in the downstream (as the bottleneck queuing occurs in the service provider's equipment, such as a Digital Subscriber Line Access Multiplexer, DSLAM).

Latency-tolerant applications often utilise the Transmission Control Protocol (TCP) to provide a reliable end-to-end data transport that adapts to available network capacity between sender and receiver [3]. TCP's congestion control (CC) algorithms have evolved over time, usually to provide improved performance in terms of usable throughput (or goodput) for high speed network connections; one notable example of a new, experimental TCP variant is CUBIC [4], which is the default mechanism in Linux at the time of writing. What all currently deployed TCP CC algorithms have in common is that they cause a cyclical filling and draining of the queues at the path's bottleneck link. This in turn induces fluctuating latency and periodic packet loss to all traffic sharing the link.

Consumers with fractional- and multiple-Mbps "broadband" links to the Internet often find their home broadband connection is the bottleneck link for individual communication sessions (for example, the data rates of ADSL 1 or ADSL 2 are dwarfed by today's 100 Mbps and 1 Gbps home LANs and service providers' edge network speeds). It would then seem natural to give users full control over what happens on their home connection — but in reality, such control is quite limited, and where it is available, there is a lack of guidance. By showing the impact of a normally "invisible" parameter (a detail in the drop behavior of the simple, and most commonly used, FIFO queue), and comparing the likely collateral damage caused by NewReno [5] and CUBIC

[1]Latencies of 100 ms, 500 ms and 1000 ms are enough to degrade the experience for first person shooters, massively multiplayer online games and real-time strategy games, respectively [2].

Application	Payload size (bytes)			Interarrival time (ms)						Average bandwidth	
	avg	min	max	avg	med	min	max	1%	99%	(pps)	(bps)
Windows remote desktop	111	8	1417	318	159	1	12254	2	3892	3.145	4497
VNC (from client)	8	1	106	34	8	< 1	5451	< 1	517	29.412	17K
G.711 RTP VoIP	180	180	180	20	20	19	21	19	21	50	64K
Skype (2 users) (UDP)	111	11	316	30	24	< 1	20015	18	44	33.333	37K
Skype (2 users) (TCP)	236	14	1267	34	40	< 1	1671	4	80	29.412	69K
World of Warcraft	26	6	1228	314	133	< 1	14855	< 1	3785	3.185	2046
Quake 3 (from server)	80	40	300	50	50	30	70	35	65	20	25K

Table 1: Examples of thin stream traffic characteristics from time-dependent applications. All traces are one-way (no ACKs are recorded) packet traffic.

on interactive real-time traffic like VoIP, we intend to fill this gap. Our observations also provide cautionary guidance to researchers who model such traffic mixes, and to vendors implementing equipment at either end of consumer links.

Section 2 introduces the relevant parts of TCP behavior, and lists three superficially-similar packet dropping rules that may be encountered in live and testbed networks. Our technique for simulating and experimentally verifying some typical home broadband scenarios is described in Section 3. Section 4 presents the impact on latency of choosing CUBIC versus NewReno, and the dramatic impact on packet loss rates caused by particular definitions of ‘queue full’ events. Our paper concludes with Section 5.

2. BACKGROUND

In this section we review TCP congestion control, the types of packet drop rules one might encounter, and the importance of testing the collateral impact of TCP flows on latency-sensitive non-TCP flows.

2.1 TCP Congestion Control

The Internet’s IP-based network and underlying infrastructure has grown beyond initial architectural design assumptions and expectations in a number of areas. Since TCP congestion control (CC) was first proposed [6] and subsequently mandated [7], there has been significant ongoing research to ensure CC addressed these changes, efficiently utilising the available capacity and protecting the network from congestion collapse.

TCP dynamically adjusts its transmission speed to balance multiple competing goals – maximise the use of available network capacity, share network capacity with other users, and do not overrun the receiver with packets. A TCP sender constrains the number of unacknowledged packets in flight to the smaller of the receiver’s advertised window (*rwnd*) and the congestion window (*cwnd*). TCP receivers keep senders informed of rwnd by placing a copy in each TCP ACK (acknowledgment) packet. Assuming the receiver’s buffer is large enough, the TCP flow control window is usually constrained by cwnd.

Typically cwnd will start at a small value and grow incrementally (*additive increase*) as packets are successfully acknowledged, effectively increasing the sender’s transmission rate over time. This will also tend to increase the number of packets buffered in any bottleneck queue between sender and receiver. When a packet is detected as lost, the sender responds by reducing cwnd to some fraction of its previous value (*multiplicative decrease*) before restarting the growth cycle. TCP presumes that packet loss means a queue overflowed somewhere between sender and receiver. Dropping cwnd causes a reduction in the sender’s average transmission rate, protecting the network from congestion collapse (and allowing packets belonging to other flows to utilise the bottleneck link).

Most TCP CC schemes differ in their specific algorithms for *additive increase multiplicative decrease* (AIMD) control of cwnd and their detection of congestion. Traditional NewReno flows have trouble with wireless links (where packet losses are frequently unrelated to congestion) and paths with large bandwidth-delay product (BDP) (it can take multiple minutes to re-probe network capacity after a single packet loss [8]).

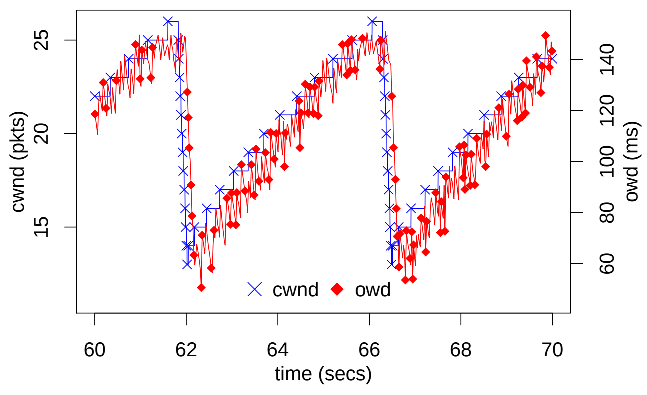
Using a simulated ADSL 1 link with a 20 000 byte bottleneck queue and 100ms baseline round trip time (RTT), Figure 1 illustrates how a single downstream flow’s cwnd fluctuation over time is cyclical and yet can be quite different depending on one’s choice of CC algorithm.

Figure 1(a) and 1(b) show a ten second sample of cwnd and induced queuing delay associated with a single NewReno and CUBIC flow respectively. The one way delay (OWD) experienced by a constant bit rate VoIP-like traffic source sharing the bottleneck link is intuitively proportional to and driven by the fluctuation of cwnd.

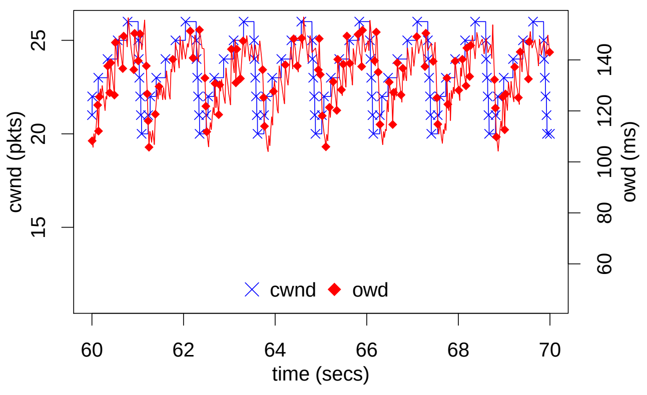
CUBIC’s aggressiveness compared to NewReno is clearly visible, both in terms of growing cwnd at a faster rate and backing off less on congestion. The lower-bound base OWD of 50ms (100ms path RTT divided by two) grows by up to 107ms to a maximum of 157ms when the queue is full. By backing off less on congestion, CUBIC does not give the queue time to drain as much as with NewReno, and therefore the queuing delay is consistently higher. Regardless of the TCP algorithm, a real VoIP session sharing this link with a TCP session would clearly experience substantial fluctuations of OWD above the path’s 50ms minimum.

2.2 Choosing when to drop packets

Buffering at bottleneck links is often implemented (and modeled in research papers) as a first-in first-out (FIFO) drop-tail queue – a newly arrived packet is dropped rather than queued if it arrives when the queue is full. Although



(a) NewReno



(b) CUBIC

Figure 1: TCP congestion window oscillation coupled with the induced queueing delay. ADSL1 type link: 1.5 Mbps/256 kbps, with a minimum 100ms RTT and 20 000 byte queue.

the notion of 'being full' might seem obvious, we identify three superficially-similar ways to define when a queue is full:

- **PS**: The queue is *packet* (*slot*) based, and the maximum queue length is specified as an integer number of packets:
```
if (NumPktsInQ ≥ PktQueueLength)
   drop incoming packet
```
- **FB-strict**: A *fixed* length *byte*-based queue where the current queue length + new packet length must not exceed a specified number of bytes:
```
if ((NumBytesInQ + InPktLength)
                              > ByteQueueLength)
   drop incoming packet
```
- **FB-loose**: A *fixed* length *byte*-based queue where the current queue length must not exceed a fixed number of bytes before adding the new packet:
```
if (NumBytesInQ > ByteQueueLength)
   drop incoming packet
```

Network devices, simulators and emulators may implement one or more of the preceding packet drop rules, without necessarily making the actual rule clear to their users. For example, FreeBSD's *dummynet* [9] (a tool used by many researchers to emulate various network conditions in small testbeds[2]) uses PS by default but silently switches to FB-loose if the queue size is configured in bytes. Commercial equipment and network simulators (such as ns-2 [10]) often implement PS or FB-strict.

Our experiments reveal that one's choice of PS, FB-strict or FB-loose packet drop rule can have significantly different impacts on what we predict or expect to see when TCP and non-reactive flows share a bottleneck.

2.3 Interactions between application flows

Community evaluation of new TCP CC schemes has tended to focus on aspects such as intra-protocol fairness, inter-protocol fairness with NewReno, maximising throughput and speeding convergence to a fair transmission rate [11]. Although broad consensus has yet to be reached, NewReno has already been replaced by CUBIC as the default CC algorithm in recent releases of Linux, and Microsoft is migrating to the use of *CompoundTCP* [12].

However, most consumers will experience a mixture of entertainment, information access and communication services over a single (often asymmetric) broadband IP service to their homes. To date, we have found very little prior work [13, 14, 15] investigating home broadband scenarios and the behaviour of emerging TCPs, or TCPs interacting with non-congestion reactive, latency-sensitive traffic in this environment. The continuing convergence towards IP based entertainment, information access and communication service delivery ensures this is an area of increasing importance. Consequently it is crucial to properly model and understand the likely behavior of different TCP CC schemes when interacting with non-congestion reactive, latency-sensitive traffic in such a consumer environment.

Online multiplayer games and VoIP are two examples of latency-sensitive consumer applications that do not typically react to congestion in the network. Online multiplayer computer games, and the popular first person shooter genre in particular, have well known latency sensitivity requirements for effective game play [16]. Voice telephony is another real-time service with well characterised latency tolerances (see [17, 18]). Markopoulou et al. [19] look at these performance issues over IP backbone. Such applications typically generate relatively constant streams of UDP packets in the 80 to 300+ byte range (small in comparison to typical TCP data packets).

In the rest of this paper we explore how such applications are likely to be impacted by sharing a home broadband link with NewReno and CUBIC TCP flows. In particular we focus on the case where TCP-based content is being delivered into a home, cyclically congesting the downstream link shared with a VoIP-like flow.

3. EVALUATION METHODOLOGY

Our evaluation follows on from our previous work in [15] by exploring a significantly wider range of more realistic scenarios including asymmetric bandwidth and variable queue

[2]http://citeseerx.ist.psu.edu/ lists 311 citations (accessed 13 April 2010) of [9] for dummynet

size. We utilise both an experimental testbed (Figure 2) and ns-2 [10] simulating the same network topology. The testbed's FreeBSD router uses dummynet [9] to emulate the bottleneck drop-tail queues and RTT/2 of delay in each direction[3]. Hosts A (TCP sender) and C (TCP receiver) run Debian Linux[4]. The other communicating endpoints (hosts B and D) run FreeBSD 7.0.

While not topologically equivalent to real network paths, our simple *dumbbell* testbed focuses on emulating a consumer's asymmetric ADSL-style bottleneck link combined with realistic end-to-end RTT characteristics.

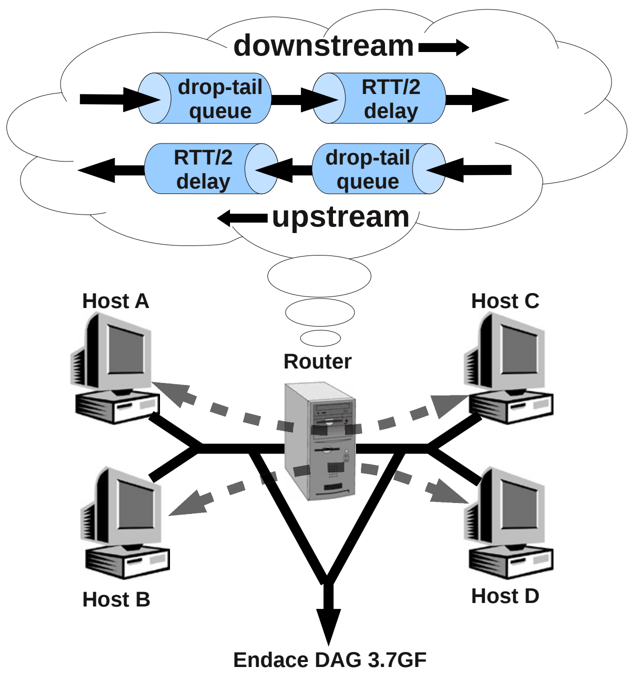


Figure 2: Testbed for investigation of TCP interaction with non-reactive CBR traffic

For clarity we evaluated the interaction between a single downstream TCP flow and a bi-directional non-reactive constant bit rate (CBR) flow under the following scenarios:

- Typical consumer ADSL 1 and high-end ADSL 2 speeds of 1500/256 kbps and 24/1 Mbps respectively.
- Byte-based queues of length 10 000, 20 000, 40 000, 60 000, and 100 000 bytes[5]
- Slot-based queues of length 7, 14, 20, 27, 34, 40, 47, 54, 60 and 67 slots
- Round trip time (RTT) delays of 24, 50, 100, and 200 ms.
- PS and FB-loose packet dropping in the testbed
- PS, FB-strict and FB-loose in ns-2[6].

For PS packet dropping the slot-based queue limit was calculated as the number of 1500 byte packets that fit into the equivalent byte-based queue lengths, rounded up to the nearest slot.

3.1 Testbed traffic generation

Bulk TCP traffic was generated using Iperf [22], with data flowing from Host A to C and ACKs from Host C to A. Non-reactive CBR UDP traffic was emulated using tcpreplay[23] to send two uni-directional streams of 186 byte UDP packets from Hosts B to D and D to B. The UDP packets are sent on average every 20 ms, with a normally distributed jitter with a standard deviation of 1 ms.

3.2 Measurements

Trials were run five times for each combination of TCP algorithm, ADSL speed, queue size, and RTT. Trials lasted for at least five minutes. Where applicable, graphs plot the median with error bars spanning the range of results from the repeated trials.

We used Web100 [24] on testbed Hosts A and C to enable polling of *cwnd* every 1ms over the lifetime of each TCP session. Two Endace DAG 3.7GF gigabit Ethernet capture ports were used for precision traffic capture to calculate delays through the router.

4. RESULTS

For home users, TCP performance is not substantially impacted by the choice of NewReno or CUBIC. However, CUBIC induces noticably higher additional latency than NewReno, and the FB-loose packet drop scheme creates much higher packet losses than PS or FB-strict.

4.1 TCP Goodput

A useful indication of TCP performance is *goodput* – the usable throughput experienced by an application (i.e. not counting retransmitted packets). As noted in Figure 1, cwnd cycles as TCP probes for available capacity and regularly hits a bottleneck queue's limit. Larger queues lead to more time with larger cwnd, and hence higher goodput.

Figure 3(a) shows TCP goodput for various bottleneck queue sizes (in units of 1500 byte packets) for a 1.5Mbps/256kbps link and 100ms base RTT. CUBIC was able to fully utilise the bandwidth with the smallest tested queue size of 7 packets, while NewReno required 20 packets.

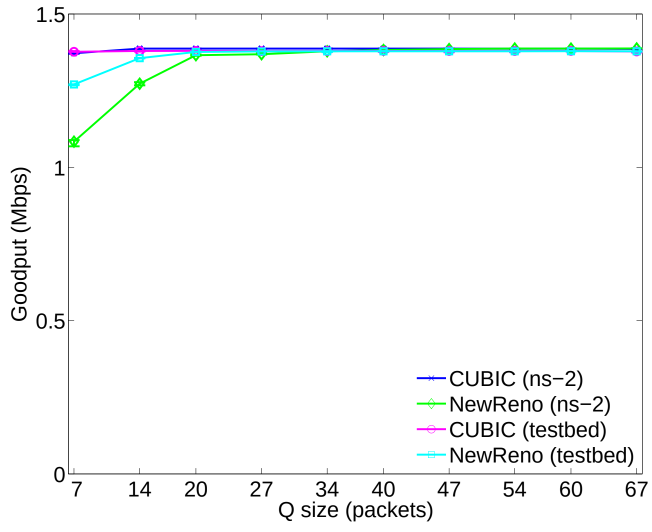
Figure 3(b) shows goodput for a 24/1 Mbps link and 100ms base RTT. CUBIC and NewReno both fail to fully utilise the bandwidth at queue sizes below 40 packets. NewReno was unable to drive the link at full capacity even with the largest queue size we tested with. It follows in the subsequent sections that the marginally better goodput achieved by CUBIC comes at a price. In our view, that cost outweighs the benefit in a home environment, where absolute performance is less important than the full range of services in

[3]Configured latency is accurate to within 0.5ms as the router's kernel was set to tick at 2000Hz ($kern.hz = 2000$). FreeBSD 7.0-RC1 on a 2.80 GHz Intel Celeron D (256K L2 Cache), 512 MB PC3200 DDR-400 RAM, with two Intel PRO/1000 GT 82541PI PCI gigabit Ethernet cards as forwarding interfaces

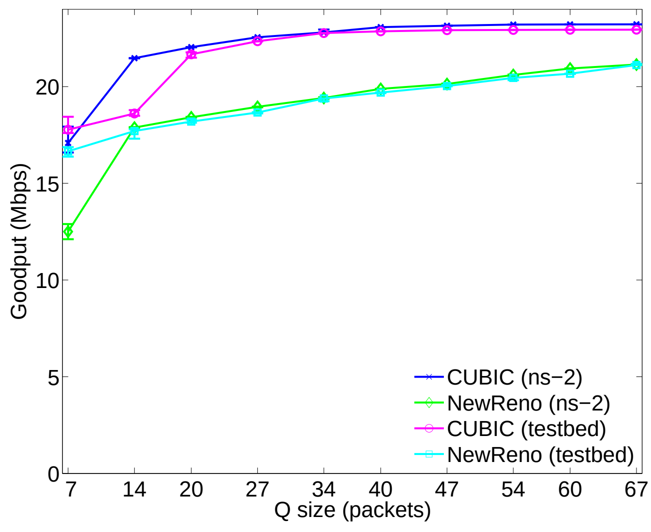
[4]A 2.6.25 kernel ticking at 1000 Hz, each one a 1.86 GHz Intel Core2 Duo E6320 (4 MB L2 Cache) CPU, 1 GB PC5300 DDR2 RAM and Intel PRO/1000 GT 82541PI PCI gigabit Ethernet NIC. Load-time tunable variables of the Linux CUBIC implementation were left at their default values.

[5]Sizes are based on previously published estimations of buffering in consumer routers [20, 21]

[6]ns-2, version 2.33 had to be modified slightly to ensure the FB-strict implementation properly accounted for the length of the packet in the front of the queue.



(a) ADSL1 TCP Goodput – 1.5 Mbps/256 kbps



(b) ADSL2 TCP Goodput – 24 Mbps/1 Mbps

Figure 3: Downstream TCP goodput vs PS queue length – measured (testbed) and simulated (ns-2) for a 1.5 Mbps/256 kbps link and 24/1 Mbps link, 100 ms minimum RTT.

use working adequately "out of the box" without specialised configuration.

The testbed and ns-2 simulations compare well, except when the buffer size is very small. Small buffer sizes tend to magnify the discrepancies between the ns-2 model and the real Linux TCP stack, with the cwnd increase and backoff oscillations occuring more rapidly on the testbed (see section 4.2.1 for more discussion).

4.2 CBR packet loss: results of trial runs

The proportion of CBR packets lost due to bottleneck queue congestion is very sensitive to the choice of queue full packet drop mechanism (Section 2.2). Figure 4(a) shows the proportion of CBR packets lost on the downstream of a 1500/256 kbps bottleneck link, assuming the TCP flow experiences a minimum RTT of 100ms.

4.2.1 FB-loose

FB-loose allows overfilling by at most one packet. Since TCP data packets are typically much larger than a VoIP packet and arrive at the queue more frequently, the TCP data packet is more likely to overfill the queue (and when this happens the queue remains full for longer). Figure 4(a) illustrates that FB-loose consequently introduces much higher CBR loss rate than PS or FB-strict.

Figure 4(b) shows that our NewReno simulation and testbed results are very close, although the simulation results tend to underestimate real CBR packet loss for CUBIC. (Wei and Cao [25] find that the ns-2 CUBIC cwnd oscillation and congestion epochs differ in their own Linux/dummynet testbed trials. The ns-2 CUBIC model has less congestion epochs than the real Linux implementation, resulting in a smaller number of CBR packets being lost.)

Although the general CBR loss rate trends down as the queue size increases, it also depends on how the actual packet sizes fit into the queue. Both the testbed and ns-2 simulations highlight this especially for CUBIC in Figure 4(b) for queue sizes which are multiples of TCP's 1500 byte packets.

There is a stark difference in the proportion of CBR packets lost between FB-loose and FB-strict (explained in section 4.3). This should be of interest to researchers using dummynet to emulate typical in-network queues, as dummynet's use of FB-loose may not be representative of FB-strict implementations in deployed equipment.

4.2.2 FB-strict

When a FB-strict queue approaches full there can be room at the end for one or more of our 186 byte CBR packets even when there is no room for a 1500 byte TCP data packet. This leads to a bias toward accepting smaller packets when the link's capacity is primarily being consumed by large TCP packets. As shown in Figure 4(a), FB-strict saw almost 0 % loss of CBR packets. ISPs who configure their down link queues to be byte based with the FB-strict drop policy can expect to provide their customers with better VoIP or online game play experience in terms of minimising packet loss.

4.2.3 PS

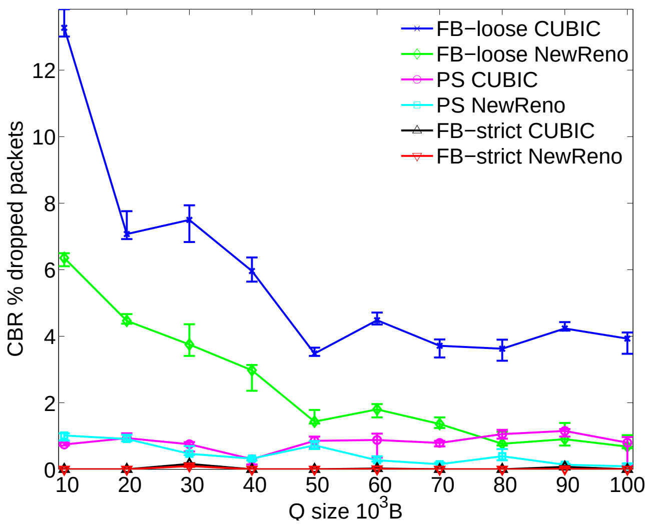
PS treats all packets equally, regardless of size, resulting in a moderate CBR packet loss. This queue drop mechanism also provides a degree of robustness to the overall fairness between reactive and non-reactive flows when the non-reactive flows have much smaller packets than the reactive flows (though this may well be a non-goal).

4.2.4 ADSL 2

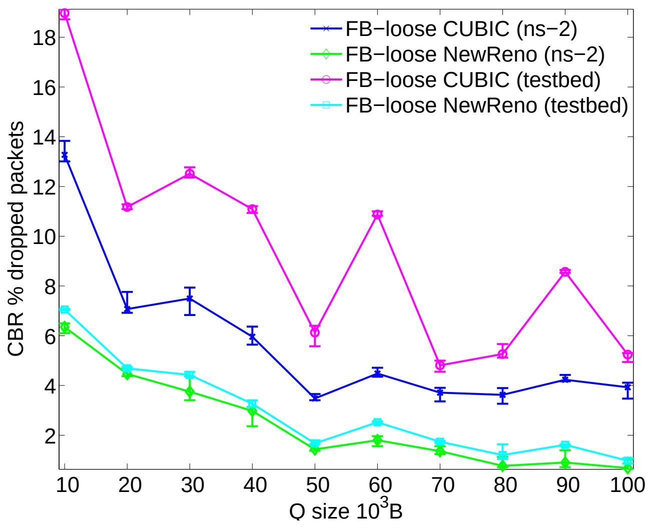
Our equivalents of Figures 4(a) and 4(b) using a 24/1 Mbps link revealed that relative CBR loss rates were similarly impacted by the choice of FB-loose, PS or FB-strict packet drop scheme. However, we have not included a graph as the absolute CBR loss rates were exceedingly low (about 1 % and 0.4 % for CUBIC and NewReno respectively with FB-loose) and relatively insensitive to queue size.

4.2.5 CUBIC's Fast Convergence Heuristic

During TCP's congestion avoidance phase, CUBIC uses its *fast convergence* heuristic to detect competition for bot-



(a) CBR packet loss for FB-loose, PS and FB-strict packet drop schemes (simulated with ns-2)



(b) Comparison of CBR packet loss for FB-loose – measured (testbed) and simulated (ns-2).

Figure 4: Proportion of CBR packets lost on the downstream for different packet drop mechanisms – 1500/256 kbps link, 100ms minimum RTT.

tleneck resources and more aggressively backoff to allow new flows to gain their share of the available bandwidth [4]. When a congestion event occurs, the heuristic compares the value of cwnd from the previous congestion event against the current cwnd and backs off more aggressively if the current cwnd is smaller. The reasoning for this behaviour is that if a new flow begins competing for buffer space, the buffer will fill sooner and our flow's cwnd will not have grown as much when the next congestion event occurs.

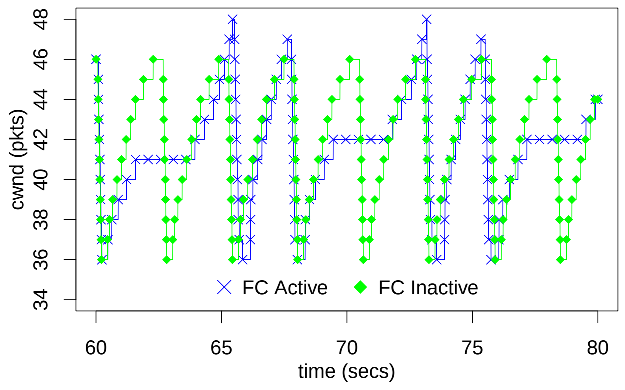


Figure 5: Comparison of CUBIC flows from separate but identical trials showing the effect of the fast convergence heuristic – single CUBIC TCP vs single CBR flow, 50ms RTT, 1.5Mbps/256kbps link, 60000 byte queue.

We frequently observed single flow CUBIC trials operating in either *fast convergence* or regular congestion avoidance mode exclusively. All other parameters being the same, operating exclusively in the *fast convergence* mode results in fewer congestion epochs compared to operating exclusively in regular congestion avoidance mode. Figure 5 shows the phenomenon using representative twenty second samples from two otherwise identical, three minute long trials. The "FC Active" flow became synchronised with the CBR flow after the first congestion event in such a way that it continually triggered the fast convergence heuristic. In contrast, the "FC Inactive" flow reached the same cwnd value every congestion event and never triggered the heuristic.

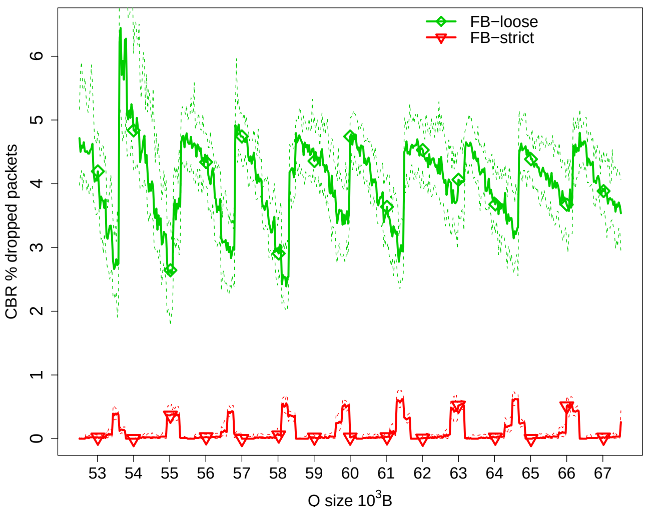
Fewer congestion epochs induces lower CBR loss rates, making comparison of data between such tests misleading. By jittering the interarrival times of the CBR UDP packets using a normal distribution (mean: 20ms, stddev: 1ms), we reduced the probability of the phenomenon occurring to the point where we did not observe it in any of our subsequent tests.

It is unlikely that this phenomenon would be induced by flows interacting in the wider Internet where jitter is naturally added by shared networking devices along a path. However, the potential for it to occur in home networks where there is often very little cross-traffic and small numbers of TCP flows sharing a bottleneck link requires further investigation.

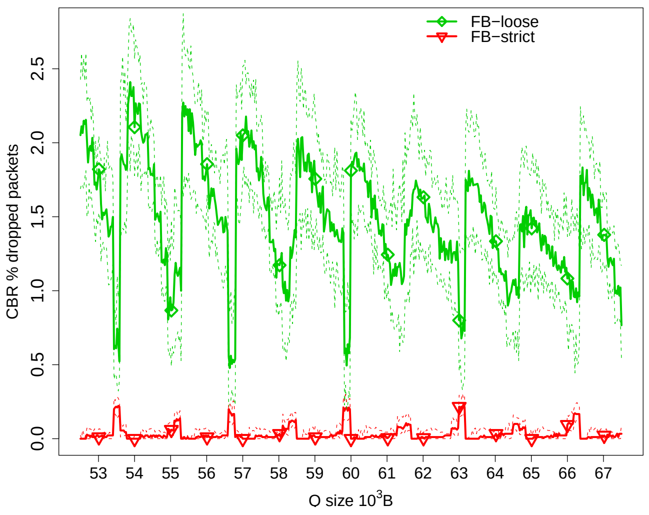
4.3 CBR packet loss: analysis

The significant difference in loss between the PS queue and either of the byte-based queues in the previous section is to be expected because of the difference in TCP and CBR packet sizes. What is surprising is the large difference in loss produced by the two different byte based queue drop rules and counter-intuitive relationship between loss rate and queue size (the peaks around queue sizes 30, 60 and 90 kbyte in Figure 4(b)).

When a queue is close to full, a 1500 byte TCP packet will have a significantly smaller chance to fit in than a 186 byte CBR packet. CBR packets arrive at the queue relatively infrequently (based on the VoIP encoding clock), while the TCP packets dominate the queue at times of congestion because of TCP's bandwidth probing behaviour. For the following discussion, we therefore assume that there are only TCP packets and no CBR packets in the queue (the exis-

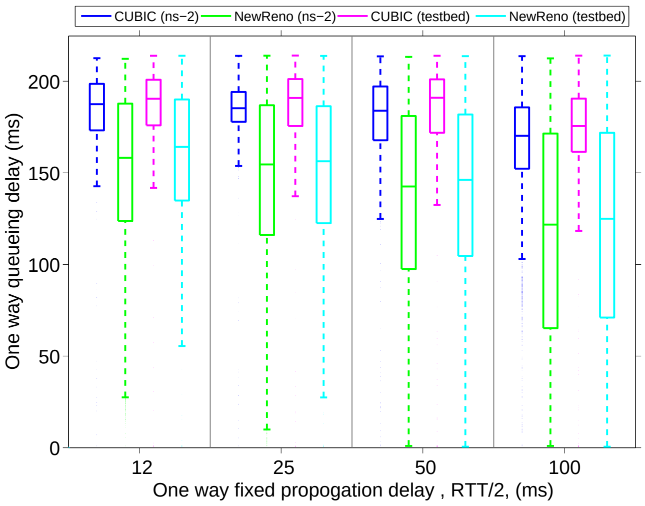


(a) CBR packet loss for FB-loose and FB-strict queues. TCP stream uses CUBIC CC.

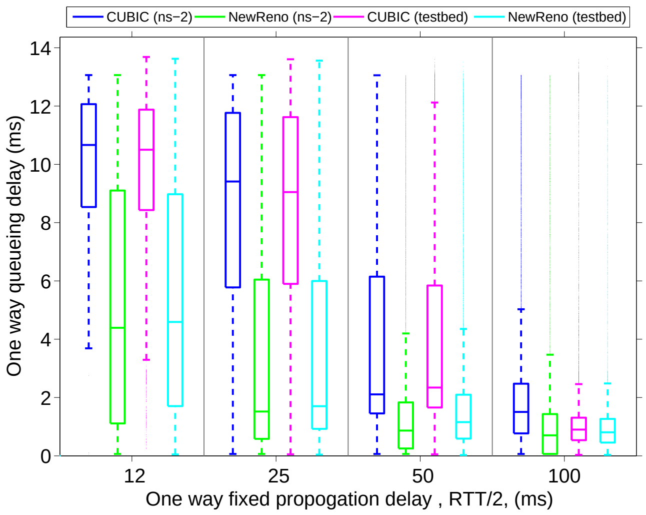


(b) CBR packet loss for FB-loose and FB-strict queues. TCP stream uses NewReno CC.

Figure 6: More detailed simulation based on Figure 4, with queue sizes ranging from 52500 bytes to 67500 bytes – 1500/256 kbps link, 100ms minimum RTT, 30 byte queue increments. The dashed line envelope represents the 5% and 95% confidence intervals for the plot data.



(a) 1.5Mbps/256kbps



(b) 24Mbps/1Mbps

Figure 7: Additional downstream CBR queuing delay experienced on a link shared with a CUBIC or NewReno TCP flow – measured (testbed) and simulated (ns-2), 40 000 byte FB-loose queue.

tence of CBR packets in the queue, the probability of which depends on the queue length, would modulate the numbers discussed below but leave the described effect intact).

In an almost full FB-strict queue, TCP packets will be dropped most of the time, while there will usually be room for a CBR packet, e.g. a queue of size $(n \times 1500 + 1499)$ bytes would always leave room for $int(1499/186) = 8$ CBR packets, irrespective of the value of n and the amount of greedy traffic which arrives. This behaviour makes the FB-strict queue friendlier towards the CBR traffic in most configurations of queue size. Based on the dynamics and parameters of a particular test, we expect there to be a range of queue sizes that, when unable to accept additional TCP packets, will also have insufficient room to accommodate any extra CBR packets. We would expect the range to be about 186 bytes (size of a CBR packet) and expect a queue of such size to induce higher CBR losses than usual.

In an almost full FB-loose queue, the ability to overfill the queue shifts the friendliness of the scheme towards the TCP sender. Since TCP packets flow at a much faster rate than CBR packets, the chance that a TCP packet will overfill the queue is greater. The increased probability of a large TCP packet overfilling the queue results in higher CBR loss waiting for the queue to drain. By allowing the queue to overflow by an entire packet, the behaviour of an FB-loose queue is very dependent on the exact contents of the queue and timing between flows. The interaction of CBR and TCP interarrival times with the path RTT determines if the queue overflows by a small or large amount.

We performed an additional set of simulations to explore these queue effects more closely. Thirty, 180 second trials were run for each permutation of TCP (CUBIC and NewReno), FB queue-drop type (FB-loose and FB-strict) and queue size (52500 to 67500 bytes in 30 byte increments).

Figure 6 plots the results, with each point on each line representing the median of all thirty trials for that permutation. The 5% and 95% confidence intervals are given by the dashed line envelope around the median value plot. Figure 6 is effectively zooming in on Figure 4(a) and providing more visibility between the coarsely-grained data points.

Starting with the FB-strict plot, the periodic peaks correspond with queue sizes in the 186 byte "sweet spot" that cause significantly increased CBR loss. For this set of trial parameters, the peaks occur at queue sizes just below an integer multiple of 1500 byte packets (e.g. 57000, 58500, 60000 bytes).

In contrast, the FB-loose plot's minima are complex and not amenable to being described by a simple mathematical model. Under the conditions shown in Figure 6, the queue overfills by a large (1500 bytes worst case) amount at queue sizes slightly larger than an integer multiple of the TCP packet size. This observation directly relates to the CBR vs TCP occupancy of the queue at the time of congestion. Changing the RTT or intearrival times would shift the plot along the x-axis, changing the locations of the loss minima.

The behaviour demonstrated in Figure 6 clearly accounts for the counter-intuitive FB-loose CBR loss figures seen in Figure 4.

4.4 CBR latency

The latency experienced by CBR packets is important for real-time applications such as VoIP and online games. Figure 7 uses box-and-whisker plots to summarise the additional queuing delay experienced by CBR packets on the downstream (total one way delay is queuing delay plus RTT/2). Box-and-whisker plots are shown for CUBIC and NewReno (ns-2 simulation and testbed) for each minimum RTT/2 (minimum one way delay) used in the experiments, with a queue size of 40×10^3 bytes.

The spread of delays experienced by CBR packets is driven by the TCP CC sawtooth, which in turn depends on the RTT – how fast it can probe, and how long it takes for TCP to react to congestion. As the propagation delay increases, TCP's probing and response to congestion becomes slower, and the variation in delay increases. Notice that CUBIC's aggressive congestion control ensures that the bottleneck queue stays fuller, resulting in a higher median latency and tighter variance than NewReno.

At 24/1 Mbps link speed the total additional latency is lower, but CUBIC still induces noticeably higher latency than NewReno. The maximum additional queuing delay is about 13 ms. Median delay introduced by CUBIC and NewReno is about 11 ms and 5 ms respectively when RTT/2 is 12 ms, and 9 ms and 1 ms respectively when RTT/2 is 25 ms. (As RTT increases further neither CUBIC nor NewReno could saturate the link, resulting in median additional delays under 2 ms for both.)

5. CONCLUSION

We have specifically considered the impact of reactive TCP-based data transfers on non-reactive real-time traffic (such as VoIP) sharing the downstream of an asymmetric consumer broadband link. Modifications to the traditional NewReno TCP congestion control algorithm to better utilise high bandwidth links can noticeably increase the typical latencies experienced by other traffic (particularly at common ADSL 1 link speeds). Second, superficially-similar packet dropping rules on the consumer and ISP sides of a home Internet connection can induce distinctly different packet loss rates in non-reactive real-time type traffic. These increases in loss and latency significantly degrade the quality of service of non-reactive real-time services.

Our observations suggest that a less aggressive form of TCP, such as NewReno, may be a better choice for delivering content toward consumers than CUBIC if one wishes to minimise additional latency induced by TCP traffic. We also observe that both researchers and implementors must pay close attention to how their simulation of a network device defines a queue being 'full'. In particular, fixed byte-length queues that allow themselves to overfill by 'one more packet' can induce substantially higher packet loss rates than queues bounded (regardless of packet lengths) by strict byte lengths or integer numbers of packets. Finally, we believe our observations may be applied to selection of packet drop mechanisms on both the upstream and downstream sides of consumer broadband modems.

Acknowledgment

This work has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

6. REFERENCES

[1] A. Petlund, K. Evensen, C. Griwodz, and P. Halvorsen, "Tcp mechanisms for improving the user

experience for time-dependent thin-stream applications," in *33rd IEEE Conference on Local Computer Networks, 2008.*, oct. 2008, pp. 176 –183.

[2] G. Armitage, M. Claypool, and P. Branch, *"Networking and Online Games - Understanding and Engineering Multiplayer Internet Games"*. UK: John Wiley & Sons, April 2006.

[3] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control ," RFC 2581 (Proposed Standard), Apr. 1999, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[4] L. X. I. Rhee and S. Ha, "CUBIC for fast long-distance networks," North Carolina State University, Tech. Rep., Aug. 2008. [Online]. Available: http://tools.ietf.org/id/draft-rhee-tcpm-cubic-02.txt

[5] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 3782 (Proposed Standard), Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3782.txt

[6] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols.* New York, NY, USA: ACM, 1988, pp. 314–329.

[7] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (Standard), Oct. 1989, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[8] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649 (Experimental), Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[9] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[10] "The network simulator - ns-2," accessed 19 Nov 2007. [Online]. Available: http://www.isi.edu/nsnam/ns/

[11] L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, and I. Rhee, "Towards a common TCP evauation suite," in *Sixth International Workshop on Protocols for Fast Long-Distance Networks*, Manchester, GB, Mar. 2008.

[12] D. B. M. Sridharan, K. Tan and D. Thaler, "Compound TCP: A new TCP congestion control for high-speed and long distance networks," Microsoft, Tech. Rep., Nov. 2008. [Online]. Available: http://www.ietf.org/internet-drafts/draft-sridharan-tcpm-ctcp-02.txt

[13] L. Andrew, I. Atov, D. Kennedy, and B. Wydrowski, "Evaluation of FAST TCP on low-speed DOCSIS-based access networks," in *IEEE TENCON 05*, Melbourne, Australia, Nov. 2005.

[14] G. Armitage, L. Stewart, M. Welzl, and J. Healy, "An independent H-TCP implementation under FreeBSD 7.0: Description and observed behaviour," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 27–38, 2008.

[15] L. Stewart, G. Armitage, and A. Huebner, "Collateral damage: The impact of optimised TCP variants on real-time traffic latency in consumer broadband environments," in *Proceedings of IFIP/TC6 NETWORKING 2009*, Aachen, Germany, May 2009.

[16] G. Armitage, "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3," in *11th IEEE International Conference on Networks (ICON 2003)*, Sydney, Australia, Sep. 2003, pp. 137–141.

[17] G. K. Helder, "Customer evaluation of telephone circuits with transmission delay," *Bell System Technical Journal*, vol. 45, pp. 1157–1191, Sep. 1966.

[18] N. Kitawaki and K. Itoh, "Pure delay effects on speech quality in telecommunications," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 4, pp. 586–593, May 1991.

[19] A. Markopoulou, F. Tobagi, and M. Karam, "Assessing the quality of voice communications over internet backbones," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 5, pp. 747–760, Oct. 2003.

[20] M. Claypool, R. Kinicki, M. Li, J. Nichols, and H. Wu, "Inferring queue sizes in access networks by active measurement," in *Passive and Active Measurement Workshop*, Antibes Juan-les-Pins, France, Apr. 2004. [Online]. Available: http://www.pamconf.org/2004/papers/209.pdf

[21] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing residential broadband networks," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement.* New York, NY, USA: ACM, 2007, pp. 43–56.

[22] "Iperf - the TCP/UDP bandwidth measurement tool," May 2005, accessed 19 Nov 2007. [Online]. Available: http://dast.nlanr.net/Projects/Iperf/

[23] A. Turner, "Tcpreplay," accessed 4 Dec 2008. [Online]. Available: http://tcpreplay.synfin.net/

[24] M. Mathis, J. Heffner, and R. Reddy, "Web100: extended TCP instrumentation for research, education and diagnosis," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 69–79, 2003.

[25] D. X. Wei and P. Cao, "Ns-2 tcp-linux: an ns-2 tcp implementation with congestion control algorithms from linux," in *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator.* New York, NY, USA: ACM Press, 2006, p. 9.