ROUTERS AND NETWORKS WITH NEAR-ZERO BUFFERS

Vijay Sivaraman[†], *Arun Vishwanath*[†], *Marina Thottan*[‡] *and Constantine Dovrolis*[§*]

[†]School of EE&T, University of New South Wales, Sydney, Australia
[‡]Networking Research Lab, Bell-Labs Alcatel Lucent, USA
[§]College of Computing, Georgia Institute of Technology, USA

PACS

Keywords: Internet routers, buffer sizing, near-bufferless networks, all-optical networks, anomalous loss performance, TCP, UDP

AMS Subject Classification:

*[†]vijay@unsw.edu.au, [†]arunv@ee.unsw.edu.au, [‡]marinat@alcatel-lucent.com, [§]dovrolis@cc.gatech.edu

ABSTRACT

All routers have buffers to store packets during periods of congestion. However, as Internet link speeds reach hundreds of Gigabits-per-second and beyond, it is becoming increasingly difficult to equip high-speed routers with large buffers, especially as switching moves into the all-optical domain. In this chapter we first trace the evolution in thinking over recent years on how much buffering is required at Internet routers, focusing specifically on the push towards smaller buffers, from Gigabyte down to Kilobyte sizes, making them amenable for all-optical realisation. We then highlight some of the implications of the move towards such small buffers, such as end-to-end performance for real-time and TCP traffic, the reaction of TCP to reduced buffer availability in the network, and the unexpected interactions between TCP and open-loop traffic. Finally, we propose mechanisms ranging from edge traffic conditioning to packet-level forward error correction within the network as means of overcoming the limitations posed by small buffers in the network, and speculate on the feasibility of a zero-buffer Internet core in the future.

1. INTRODUCTION

In today's network routers, buffers are used to reduce packet loss by absorbing transient bursts of traffic. They are also instrumental in keeping output links fully utilised during times of congestion. However, the increasing speed of network interfaces raises an important question concerning the size of these buffers. Under buffered routers lead to packet loss, thus adversely affecting application performance, while an over buffered router entails increased latency, complexity and cost.

We begin this chapter with a brief overview of buffer sizing, focusing on the reduction from Gigabyte to Megabyte to KiloByte buffers. Next, we examine the impact of buffer reduction on TCP traffic performance and characterise an unexpected behaviour that happens when closed-loop TCP interacts with open-loop (real-time) UDP in the KiloByte regime. Subsequently, we outline two novel solutions that are very effective to overcoming buffer limitations, namely, TCP pacing and packet-level Forward Error Correction (FEC). Finally, we summarise our contributions and conclude the chapter by envisioning a zero-buffer Internet core network in the future.

1.1. GIGABYTE BUFFERS

The widely used *rule-of-thumb*, commonly attributed to [1], suggests that the amount of buffering needed at a router's output interface is given by $B = C \times RTT$, where B is the buffer size, RTT the average round-trip time of a TCP connection flowing through the router, and C the capacity of the router's network interface. This rule-of-thumb is also called the Bandwidth Delay Product (BDP) rule. The motivation behind this rule was to guarantee 100% link utilisation. In other words, the BDP rule ensures that even when a buffer overflows, and TCP reacts by reducing its transmission rate, there are enough packets stored in the buffer to keep the output link busy, thereby ensuring that the link capacity is not wasted when TCP is increasing its transmission rate. This BDP rule was obtained experimentally using at most 8 TCP flows on a 40 Mbps core link in 1994. However, no

recommendation was made for sizing buffers when there are a significant number of TCP flows that have different RTTs.

In current electronic routers, for a typical RTT of 250 ms and capacity C of 40 Gbps, the rule-of-thumb mandates a buffer size of 1.25 GigaBytes, which poses a considerable challenge to router design. Further, the use of such large buffers (implemented using a combination of SRAM and DRAM chips) complicates router design, increases its power consumption, and makes them very expensive. The scaling and power consumption requirements for next generation routers can be successfully addressed by building and deploying optical routers in the Internet core. However, one of the primary technological limitations of optical routers is the difficulty in building large optical buffers. All-optical buffers can only be used to delay packets for about 100 ns [2]. In the case of a 40 Gbps link, this optical delay line translates to a buffer size of only a few hundred bits. It is thus worthwhile to revisit the buffer sizing problem and examine if we indeed require the amount of buffering as dictated by the BDP rule.

1.2. MEGABYTE BUFFERS

Researchers from Stanford University showed in 2004 that when a large number N of long-lived TCP flows share a bottleneck link in the core of the Internet, the absence of synchrony among the flows permits a central limit approximation of the buffer occupancy. The combined effect of multiplexing a large number of asynchronous flows leads to a buffer size $B = RTT \times C/\sqrt{N}$ to achieve near-100% link utilisation [3]. This result assumes that there are a sufficiently large number of TCP flows so that they are asynchronous and independent of each other. In addition, it assumes that the buffer size is largely governed by long-lived TCP flows only. Thus, a core router carrying $10,000$ TCP flows needs only 12.5 MB of buffering instead of 1.25 GB as governed by the earlier rule-of-thumb.

1.3. KILOBYTE BUFFERS

More recently, it has been further argued (see [4–6]) that core router buffers of the order of 20-50 packets suffice for TCP traffic to realise acceptable link capacities. The use of this model however comes at a tradeoff. Reducing buffers to only a few dozen KiloBytes can lead to a 10-20% drop in link utilisation. The model relies on the fact that TCP flows are not synchronised and network traffic is not bursty. Such a traffic scenario can happen in two ways. First, since core links operate at much higher speeds than access links, packets from the source node are automatically spread out and bursts are broken. Second, if the TCP stack running at end-hosts is altered such that it can space out packet transmissions (also called TCP pacing) [7]. The slight drop in link utilisation seems worthwhile since core links are typically overprovisioned, and it pays to sacrifice a bit of link capacity if this permits a move to either an all-optical packet switch or more efficient electronic router design.

1.4. SIZING BASED ON PER-FLOW METRICS

Researchers from Georgia Tech and Bell-Labs have tackled the buffer sizing problem from a completely different perspective [8]. Rather than assuming that most of the TCP traffic is persistent, i.e., long-lived flows that are mostly in the congestion avoidance mode, they

consider the more realistic case of non-persistent flows with flow sizes drawn from a heavy-tailed distribution. This differs from some of the prior work in that non-persistent flows may not saturate the links along their paths, unlike the persistent flows, and can remain in the slow-start phase without entering into the congestion avoidance mode. Also, the number of active flows at any instant is highly time variant. It follows that flows that spend most of their time in the slow-start phase require significantly fewer buffers than flows that spend most of their time in the congestion avoidance mode.

Further, instead of focusing purely on link utilisation, their work focuses on the average per-flow TCP throughput, which is an important metric as far as an end-user is concerned. It can be the case that a link may have sufficient buffers so that it always maintains high utilisation, but the per-flow throughput can be very low. The objective is to find the buffer size that maximises the average per flow TCP throughput. Analytical, simulation and experimental evidence are presented to suggest that the output/input capacity ratio at a router's interface largely governs the amount of buffering needed at that interface. If this ratio is greater than one, then the loss rate falls exponentially, and only a very small amount of buffering is needed. However, if the output/input capacity ratio is less than one, then the loss rate follows a power-law reduction and significant buffering is needed.

Their study concludes by pointing out that it may not be possible to derive a single universal formula to dimension buffers at any router's interface in a network. Instead, a network administrator should decide taking into account several factors such as flow size distribution, nature of TCP traffic, output/input capacity ratios, etc.

For a comprehensive understanding of the router buffer sizing problem, we refer the interested reader to our recent survey paper in [9].

2. IMPLICATIONS OF SMALL BUFFERS ON TCP DYNAMICS

There is widespread debate regarding the nature of TCP traffic in today's Internet - while some researchers have shown that it exhibits long-range dependent (LRD) properties, others argue that it can be modelled as a Poisson process due to the high degree of traffic aggregation that exists in the core. In this section, we investigate the nature of TCP traffic as the Internet core moves towards an all-optical packet network with very limited buffering (few tens of KiloBytes) capability.

The publication of the seminal paper in [10] showed empirically that network traffic exhibits long-range dependent characteristics (i.e., bursty traffic over a wide range of time-scales). The heavy-tailed distribution of the file transfer sizes and the duration of active/inactive times of users are the two main reasons why network traffic is thought to be LRD. Since the publication of this result, there has been a keen interest in trying to understand how Internet traffic has evolved. Many have argued that due to the phenomenal growth in Internet traffic [11] (currently in the exabyte (10^{18}) range) and link speeds, there is high degree of statistical multiplexing and traffic aggregation in the core, resulting in the traffic being Poisson in sub-second time-scales [12]. There is further support from stochastic processes theory, which suggests that the superposition of a large number of *independent* flows also converges to a Poisson process [13–15].

Contrary to the above claims, there is still a large body of work showing that despite the growth in the Internet, the traffic is still bursty (LRD) and not smooth (Poisson). Through a combination of wavelet-based analysis and trace data obtained from a OC-48 Internet2 core link, the authors of [16] showed that Internet traffic is bursty in short time-scales because of the self-clocking nature of TCP and queueing at the bottleneck link. Also, as previously argued in [12], the aggregation of many TCP flows does not make the traffic smooth. Further light is shed in the recent paper [17] that uses trace data (spanning seven years) from a trans-Pacific backbone link to show that Internet traffic is indeed LRD and not Poisson.

It must be mentioned that the above measurement studies are pertinent to today's core links that have of the order of delay-bandwidth amount of buffering, since ISPs typically run their networks with such large buffers [1]. Although the recent work advocating the use of small buffers has assumed that TCP traffic at a bottleneck link can be modelled as being Poisson [18], a claim predominantly supported by the theoretical superposition results, to the best of our knowledge, little has been done to justify this assumption using experiments and/or simulations.

The superposition results from stochastic processes are accurate only when the flows are *independent*. This is a critical assumption that easily holds for open-loop traffic (UDP for example). However, TCP is a closed-loop (i.e., feedback controlled) protocol, and the behaviour of a TCP flow is influenced by the presence of other TCP and/or UDP flows, as well as the buffer occupancy at the bottleneck link. In addition, TCP flows can synchronise their window dynamics behaviour (causing them to be in lock-step), which is a detrimental effect that breaks the independence assumption. As a result it is not very clear if the superposition result is directly applicable to TCP traffic, and as shown in [16], the aggregation of a large number TCP sources does not lend itself to being Poisson.

In the rest of this section, we show that the nature of aggregate TCP traffic is strongly dependent on the size of the bottleneck link buffer. When buffers are small, the aggregate TCP arrival process is well approximated as being Poisson, but when buffers are large (of the order of delay-bandwidth product), TCP flows tend to synchronise their congestion windows, thus creating significant bursty LRD traffic. This seems to indicate that as we move to a core optical network in the future with very limited buffering, approximating the traffic as being Poisson is well justified.

2.1. SIMULATION SETUP

In the context of studying the burstiness of TCP traffic in a small buffer core network, we require a bottleneck link carrying several TCP flows. We use *ns-2* (version 2.33) simulator on the well-known dumbbell topology shown in Fig. 1, which directly captures the bottleneck link, and is commonly used to analyse the performance of various congestion control algorithms, including TCP. We consider TCP-Reno flows corresponding to each source-destination pair *(s-tcp$_i$,d-tcp$_i$)*, $1 \leq i \leq 2000$, and employ FIFO queue with drop-tail queue management (at the bottleneck router r_0), which is commonly used in most routers today. Each TCP packet is of size 1 KB. The bottleneck link operates at 200 Mbps, while the access link speeds are uniformly distributed between $[8,12]$ Mbps (mean of 10 Mbps), to reflect a typical home user. The propagation delay on the bottleneck link is 25 ms with the access link delays being uniformly distributed in the interval $[4,10]$ ms, thus RTT varies

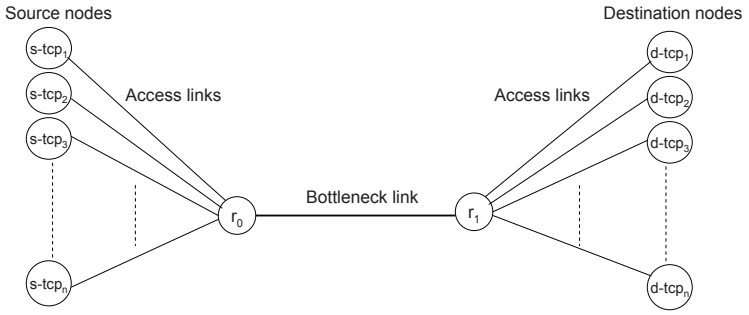


Figure 1. *ns-2* simulation dumbbell topology with TCP traffic

between $[58,70]$ ms. All TCP flows begin their transmission in the interval $[0,10]$ sec and the simulation is run for 150 sec. Data in the interval $[20,150]$ sec is used in all our computations so as to capture the steady-state behaviour of the network.

2.2. IMPACT OF BUFFER SIZE ON TCP BURSTINESS

Our objective is to test if TCP packet arrivals to the bottleneck link buffer are near-Poisson or not. To do so we measure the burstiness of the arrival traffic at various time-scales. Burstiness at time-scale s is quantified by $\beta(s)$, the coefficient of variation (i.e., ratio of standard deviation to mean) of traffic volume measured over time intervals of size s. Log-log plots of $\beta(s)$ versus s are routinely used in the literature to depict traffic burstiness over various time-scales as an indicator of long-range dependency of traffic traces, and to show the influence of the Hurst parameter H. We will use estimates of H obtained from the burstiness plots as a measure of how close the traffic is to being short-range dependent (i.e., Poisson).

Consistent with prior work [16], we are also interested in short time-scales (between $2^{10}\mu s \approx 1$ ms to $2^{20}\mu s \approx 1$ s). Time-scales of sub-1 ms is not very relevant since the average transmission time of a TCP packet on the access link is 0.8 ms (because the TCP packet size is 1 KB and the mean access link rate is 10 Mbps). Further, for a given number of TCP flows, the variation at larger time-scales is due to TCP traffic arriving and departing the network, and not because of the congestion control algorithm of TCP. Therefore, we upper-bound the time-scale of interest to 1 s, which is a few multiples of the average round-trip time.

2.3. LONG-LIVED TCP FLOWS

We begin the study by considering the simple case of long-lived TCP flows, i.e., all TCP flows are persistent and have infinite data to send.

Fig. 2 shows the traffic burstiness $\beta(s)$ as a function of the time-scale (s in μs) for six different values of bottleneck link buffer size and 2000 long-lived TCP flows. The bottom curve indicates burstiness when the bottleneck link has very limited buffering - 50 KB, equivalent to 50 packets, since each TCP packet is 1 KB in our simulation. The remaining five curves reflect increasing buffer sizes (ranging from 3 MB to 12.5 MB), with 6.25 MB of buffering corresponding to the delay-bandwidth product (average RTT of

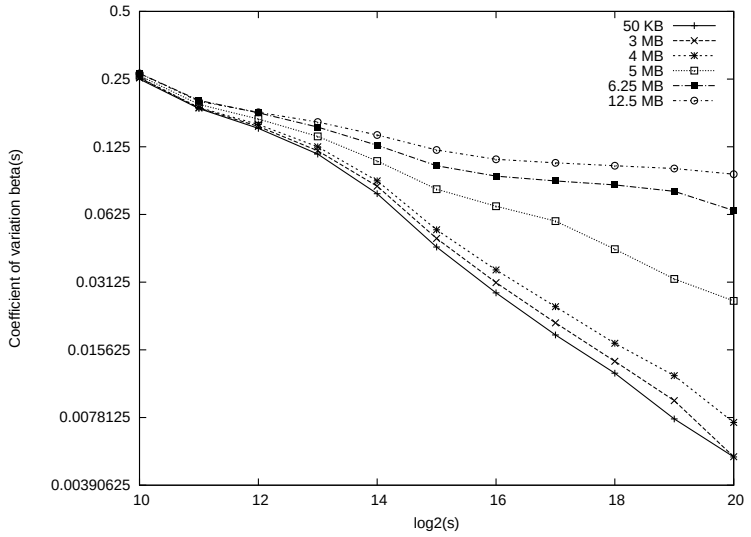


Figure 2. Burstiness for 2000 long-lived TCP flows with different buffer sizes

250 ms times 200 Mbps link rate). The curves fairly overlap each other till time-scale of $2^{12}\mu s \approx 4$ ms. However, at time-scales beyond 4 ms, the burstiness curves for large buffers (approaching the delay-bandwidth product) flatten significantly, indicating onset of long-range dependence, with the Hurst parameter estimated at approximately 0.66 (for 5 MB buffers), 0.81 (for 6.25 MB buffers) and 0.88 (for 12.5 MB buffers) respectively. These results show that when buffers are large, TCP traffic is significantly bursty and cannot be modeled as being Poisson. On the contrary, with small buffers, the Poisson approximation seems reasonable.

2.4. MIX OF SHORT-LIVED AND LONG-LIVED TCP FLOWS

Though results from the previous section indicate that TCP packet arrivals are near-Poisson if we have a large number of long-lived TCP flows and small buffers, the reader may wonder if similar behaviour can be seen when many of the TCP flows are short-lived (or equivalently, the number of TCP flows is time varying). This is an important consideration since measurement studies at the core of the Internet suggest that a large number $(> 80\%)$ of TCP flows (e.g. HTTP requests) are short-lived and carry only a small volume of traffic, while a small number of TCP flows (e.g. FTP) are long-lived and carry a large volume of traffic.

We simulate such a scenario in *ns-2* using the same dumbbell topology shown in Fig. 1. There are a total of 2000 TCP flows with 1600 of them being short-lived and the rest long-lived. These 1600 users perform successive file transfers to their respective destination nodes, with the size of the file to be transferred following a Pareto distribution with mean 100 KB and shape parameter 1.2. These chosen values are representative of Internet traffic, and comparable with measurement data. After each file transfer, the user transitions into an idle or off state. The duration of the off period is exponentially distributed with mean 1 second. It can be noted that the above traffic generation mechanism, which is a combination of several ON-OFF sources with Pareto-distributed ON periods, is long-range dependent [10].

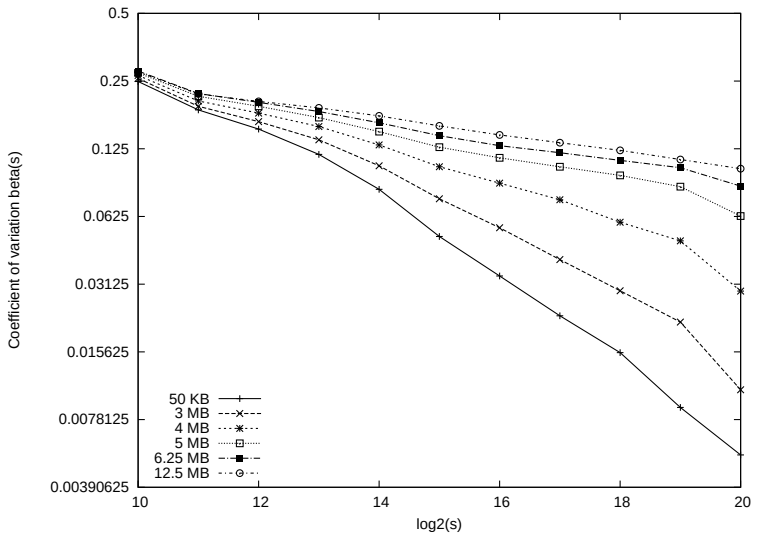


Figure 3. Burstiness for 1600 short-lived and 400 long-lived TCP flows with different buffer sizes

Fig. 3 shows the resulting burstiness curves for the same set of bottleneck link buffer sizes as before. It is interesting to note from the figure that, although a significant number of TCP flows have LRD properties, when fed into a small buffer (such as 50 KB, the bottom curve), the resulting TCP arrival process becomes near-Poisson (similar to the 50 KB curve in Fig. 2). However, we do note that as buffer size increases, the traffic tends towards LRD, as evidenced by the Hurst parameter values of 0.67 (for 4 MB), 0.79 (for 5 MB) and 0.84 (for 6.25 MB) respectively.

2.5. BURSTINESS EXPLAINED VIA QUEUE OCCUPANCY

We are now interested in understanding why the aggregate TCP arrivals in a small buffer network is near-Poisson while it is LRD with large buffers? To do so, we plot the queue (buffer) occupancy (for an arbitrarily chosen interval of 2 sec, consistent with our time-scale of interest) in Fig. 4 with 1600 short- and 400 long-lived TCP flows and two choices of buffer size - 50 KB (small buffers) and 6.25 MB (delay-bandwidth product large buffers).

We note from the figure that for large buffers, the queue spends most of its time oscillating continuously between full and $\approx 75\%$ occupancy. The reason for this oscillation is that the TCP flows are not independent, in fact they are synchronised. When the queue is almost full, many flows experience packet drops and simultaneously back-off by reducing their windows in synchrony. This causes the packets already stored in the queue to be drained out, thus providing room for the flows to ramp-up (i.e., expand their windows) again. This periodicity, and hence synchronisation, is primarily because of large buffers as TCP sources are unable to perceive loss for relatively long periods of time (due to the large queueing delays experienced by the flows). On the contrary, the queue occupancy with small buffers looks like white noise, clearly showing that small buffers are instrumental in mitigating synchronisation effects. The TCP flows can thus be treated as being independent, and the superposition results can be invoked thereby rendering credence to the fact

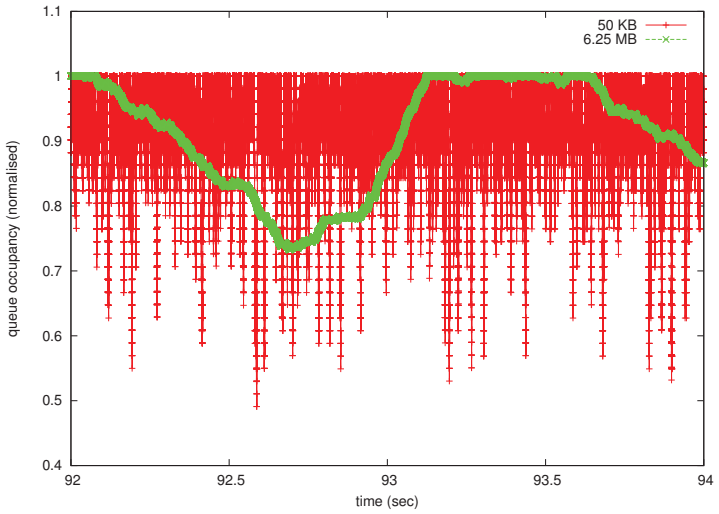


Figure 4. Normalised queue occupancy for small and large buffers with 1600 short- and 400 long-lived flows

that the aggregation of a large number of TCP flows at a bottleneck link with small buffers is near-Poisson. The synchronisation of TCP flows with large buffers is what causes them to be LRD.

2.6. SECTION SUMMARY

In this section, we showed via simulations that if we have a large number of TCP flows and small buffers, then the aggregate TCP arrival process can be well approximated as being Poisson. On the other hand, with larger buffers, the dynamics can be significantly different, and the aggregate arrival process need not converge to the Poisson model.

It would be very interesting to explore via analysis, simulations and experiments the effect of various system parameters - such as the number of TCP flows, core link speed to access link speed ratio, RTT, etc., on the burstiness of TCP. We are currently working on these directions and hope to be able to report the results soon.

3. ANOMALOUS LOSS PERFORMANCE FOR MIXED REAL-TIME AND TCP TRAFFIC

Most prior studies on buffer sizing focus entirely on TCP traffic performance, and ignore the performance implications for real-time (UDP) traffic. However, multimedia applications such as on-line gaming, interactive audio-video services, VoIP and IPTV are proliferating in the Internet, and becoming part of an ISP's revenue stream. Thus, router buffer sizing studies cannot afford to ignore the performance impact on real-time traffic when it is multiplexed with TCP, which to the best of our knowledge has not been undertaken before in the context of very small buffers. We use the term real-time and UDP interchangeably.

To understand the dynamics of buffer occupancy at a bottleneck link router with very small buffers (up to 50 KiloBytes), we mixed a small fraction of UDP traffic with TCP traffic and measured the UDP packet loss and end-to-end TCP throughput. What we observed was contrary to conventional wisdom. We found that there exists a certain continuous region of buffer sizes (typically starting from 8-10 KB or so) wherein the performance of real-time traffic degrades with increasing buffer size. In other words, packet loss for real-time traffic *increases* as the buffer size *increases* within this region. We call this region of buffer size an "anomalous region" with respect to real-time traffic.

We believe the anomalous phenomenon studied adds a new dimension to the ongoing debate on buffer sizing, including in the context of optical packet switches, and our results aid switch manufacturers and network operators in selecting small buffer sizes that achieve desired performance balance between TCP and real-time traffic.

3.1. THE ANOMALY

To illustrate the anomalous loss behaviour, we consider a simple dumbbell topology (Fig. 5) that captures packet queueing effects at the bottleneck link, which is the dominant factor in end-to-end loss and delay variation for each flow, while abstracting the rest of the flow path by an access link on either side of the bottleneck link. We use *ns*-2 [19] (version 2.30) for our simulations and consider 1000 TCP flows, corresponding to each source-destination pair *(s-tcp$_i$,d-tcp$_i$)*, $1 \leq i \leq 1000$. Further, we use TCP-Reno in all our simulations, consistent with the TCP version used in previous related work on buffer sizing, and employ FIFO queue with drop-tail queue management, which is commonly used in most routers today.

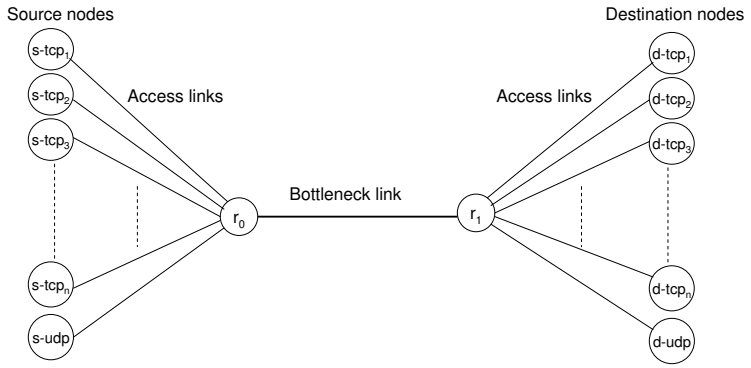


Figure 5. ns-2 simulation dumbbell topology with TCP and UDP traffic

UDP traffic is generated between nodes *(s-udp,d-udp)*. It suffices to have a single UDP flow since open-loop traffic can, without loss of generality, be aggregated. Multiple UDP flows traversing the bottleneck link can thus be modelled as a single UDP flow that represents the aggregate of all individual UDP flows passing through that bottleneck link. However, we need multiple TCP flows since they each react independently to the prevailing network condition and the state of the buffers.

The propagation delay on the UDP access link is chosen at 5 ms, while it is uniformly distributed between $[1,25]$ ms on the TCP access links. The propagation delay on the bottleneck link (r_0, r_1) is 50 ms; thus round-trip times vary between 102 ms and 150 ms. All

TCP sources start at random times between $[0,10]$ s. UDP source starts at time 0 s. The simulation duration is 800 s and performance measurements are recorded after 200 s, to allow for the stabilisation of all TCP flows.

Buffer size at the bottleneck router r_0 is varied in terms of KiloBytes. To set the packet sizes, we draw on the fact that several real-time applications, for e.g. on-line gaming applications [20] use small UDP packets since they require extremely low latencies. The study showed that almost all packets were under 200 Bytes. Our experiments using Skype and Yahoo Messenger showed that for interactive voice chat, UDP packet sizes were between 150-200 Bytes. Also, traces obtained at a trans-Pacific 150 Mbps link [21] suggests that average UDP packet sizes are smaller than average TCP packet sizes. Therefore, in all our simulations, we fix the TCP packet size at 1000 Bytes and simulate fixed and variable size UDP packets in the range of $[150,300]$ Bytes.

We first illustrate the phenomenon using the video traffic trace from the movie *Star Wars*, obtained from [22] and references therein. The mean rate is 374.4 Kbps and the peak rate is 4.446 Mbps; the peak rate to mean rate ratio being nearly 12. The packet size is fixed at 200 Bytes. We set the bottleneck link at 10 Mbps and the TCP access links at 1 Mbps, while the UDP access link is kept at 100 Mbps. The bottleneck link was only 10 Mbps because the mean rate of the video trace (UDP) is low (374.4 Kbps), and we want to keep the fraction of UDP traffic feeding into the core to within 3-10% of the bottleneck link rate (to be consistent with the nature of Internet traffic today). In this example, the video traffic constitutes $\approx 3.75\%$ of the bottleneck link rate. We have a high-speed access link for UDP since UDP traffic feeding into the core can be an aggregate of many individual UDP streams. TCP traffic on the 1 Mbps access link models traffic from a typical home user.

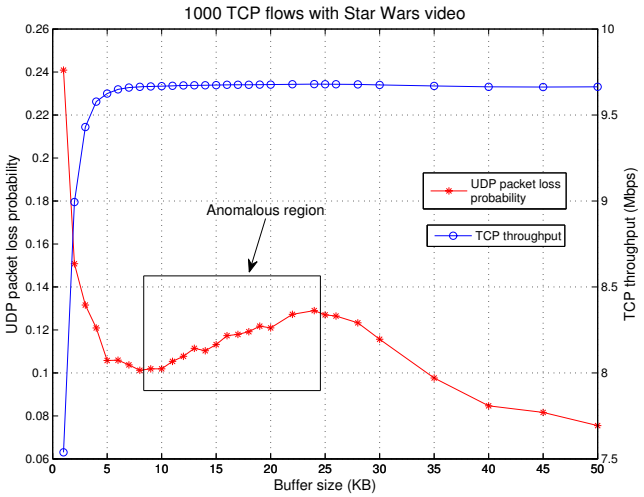


Figure 6. Starwars 200 Byte packets: UDP loss and TCP throughput

Fig. 6 shows the UDP packet loss and TCP throughput curves as a function of buffer size at the bottleneck router in the range of 1 KB to 50 KB. We see that TCP quickly ramps up to nearly 9.6 Mbps with only about 8 KB of buffering, reaching close to its saturation throughput. Simultaneously, UDP packet loss falls rapidly as well. Up to this point, both TCP and UDP behave as expected. However, as the buffer size increases beyond 8 KB till

about 24 KB, UDP performance degrades as its packet loss increases with buffer size in this region. The loss at 24 KB of buffering is approximately 30% more than the loss at 8 KB of buffering. There is however no appreciable increase in TCP throughput.

We also observed this behaviour when UDP traffic is generated from synthetic models (such as Poisson, LRD fractional Brownian motion, etc.), suggesting that the anomaly is fundamental and not just a coincidence. Moreover, in the absence of TCP traffic, UDP loss was observed to drop monotonically with buffer size, confirming that the anomaly arises due to the interaction of open and closed loop traffic at the bottleneck router with very small buffers. In the next subsections, we develop analytical models that captures the anomalous loss behaviour succinctly.

3.2. AN INTUITIVE ANALYTICAL MODEL OF THE ANOMALY

We now provide an intuitive explanation of why we think the anomaly happens, and this helps us develop a simplistic yet effective model that quantifies the buffer sharing dynamics between TCP and real-time traffic.

When buffers at the bottleneck link are extremely small, say in the range 1 KB to about 5 to 8 KB, the congestion window size of each of the TCP flows sharing the bottleneck link will also stay extremely small. TCP's congestion window is not allowed to grow beyond a few packets since several packets sent back-to-back (that would be generated by any TCP version that does not employ pacing) are more likely to be dropped at the very small buffers in the bottleneck link router. The small congestion window size implies that each TCP flow transmits only a few packets in each round-trip time, and is therefore mostly idle. Consequently, the buffers at the bottleneck link are often devoid of TCP packets, allowing UDP packets to enjoy the use of these buffers for the most part. This helps us understand why in this region, wherein TCP and UDP predominantly "time-share" the buffers, UDP loss decreases with buffer size, much like it would if TCP traffic were non-existent.

When buffer size is in the range 10-25 KB (corresponding to the anomaly), a larger fraction of the TCP flows are able to increase their congestion window (equivalently a smaller fraction of the TCP flows remain idle). This leads to higher usage of the buffers at the bottleneck link by TCP traffic, leaving a smaller fraction of the buffers for UDP traffic to use. The aggressive nature of TCP in increasing its congestion window to probe for additional bandwidth, causes the "space-sharing" of bottleneck-link buffers between TCP and UDP in this region to be skewed in favour of TCP, leaving lesser buffers available to UDP traffic even as buffer size increases.

We now try to quantify the above intuition via a simple analytical starting with the following observation. If we have a large number of TCP flows, then TCP's usage of the bottleneck buffers increases exponentially with the size of the buffer. More formally, let B denote the buffer size (in KB) at the bottleneck link, and $P_I(B)$ the probability that at an arbitrary instant of time the buffers at the bottleneck link are devoid of TCP traffic. Then,

$$P_I(B) \approx e^{-B/B^*} \tag{1}$$

where B^* is a constant (with same unit as B) dependent on system parameters such as link capacity, number of TCP flows, round-trip times, ratio of long-lived to short-lived TCP

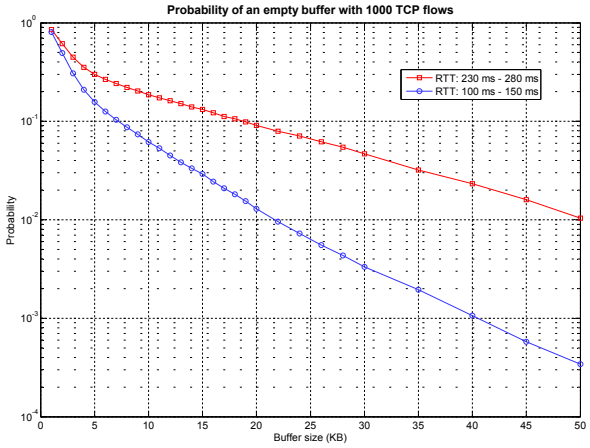


Figure 7. Probability of idle buffer vs. buffer size for TCP traffic

flows, etc. The constant B^* can be inferred from the plot of the natural logarithm of $P_I(B)$ as a function of B, which yields a straight line. The slope of the line corresponds to $-1/B^*$.

This behaviour has been observed in the past by various researchers: by direct measurement of idle buffer probabilities [23, Sec. III], as well as indirectly via measurement of TCP throughput [4, Fig. 1]: the latter has shown roughly exponential rise in TCP throughput with bottleneck buffer size, confirming that TCP's loss in throughput (which arises from an idle buffer) falls exponentially with buffer size. We also validated this via extensive simulations (shown in Fig. 6 and in various other TCP plots in later sections) in *ns-2*. 1000 TCP flows with random round-trip times from a chosen range were multiplexed at a bottleneck link, and the idle buffer probability was measured as a function of bottleneck link buffer size. The large number of flows, coupled with randomness in their round-trip times, ensures that the TCP flows do not synchronise their congestion windows. Fig. 7 plots on log-scale the idle buffer probability as a function of bottleneck buffer size for two ranges of round-trip times, and show fairly linear behaviour in the range of 5 to 50 packets (each packet was 1 KiloByte), confirming the exponential fall as per Eq. (1).

Having quantified TCP's usage of the bottleneck buffers, we now consider a small fraction f (say 5 to 10%) of real-time (UDP) traffic multiplexed with TCP traffic at the bottleneck link. The small volume of UDP traffic does not alter TCP behaviour significantly; however, TCP's usage of the buffer does significantly impact loss for UDP traffic. If we assume the buffer is very small (a few tens of KiloBytes), we can approximate the buffer as being in one of two states: idle (empty) or busy (full). With the objective of estimating the "effective" buffers space available to UDP traffic, we identify the following two components:

- **Fair-share:** During periods of time when TCP and UDP packets co-exist in the buffer, the buffer capacity B is shared by them in proportion to their respective rates. The first-in-first-out nature of service implies that the average time spent by a packet in the system is independent of whether the packet is UDP or TCP, and Little's law can be invoked to infer that the average number of waiting packets of a class is proportional to the arrival rate of that class. UDP packets therefore have on average

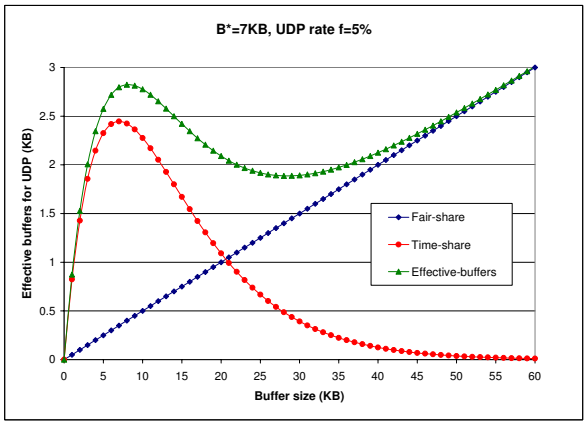


Figure 8. Effective buffers for UDP traffic

access to a "fair share" of the buffers, namely fB, where f denotes the fraction of total traffic that is UDP.

- **Time-share:** Whenever the buffer is devoid of TCP traffic (i.e. with probability $P_I(B)$), UDP packets have access to the remaining buffer space $(1-f)B$ as well. We call this the "time share" portion, since this portion of the buffer is shared in time between UDP and TCP traffic. The time-share portion of buffers available to UDP is therefore $P_I(B)(1-f)B$.

Combining the fair-share and time-share components, and invoking Eq. (1) gives us an estimate of the total "effective" buffers $\bar{B}^{udp}$ available to UDP traffic:

$$\bar{B}^{udp} = fB + (1-f)Be^{-B/B^*} \tag{2}$$

To illustrate the significance of this equation we plot it for $f = 0.05$ (i.e. 5% UDP traffic) and $B^* = 7$ KB (consistent from Fig. 7). Fig. 8 shows the total effective buffers for UDP, as well as the fair-share and time-share components. The fair-share component fB increases linearly with buffer size, while the time-share component $(1-f)Be^{-B/B^*}$ rises to a peak and then falls again (readers may notice a shape similar to the Aloha protocol's throughput curve): this happens because smaller buffers are more available for UDP to time-share, but as buffers get larger TCP permits exponentially diminishing opportunity for time-sharing. The total effective buffers for UDP, being the sum of the above two components, can therefore show anomalous behaviour, i.e., a region where larger real buffers can yield smaller effective buffers for UDP. For any realistic UDP traffic model (note that our analytical model does not make any specific assumption about the UDP traffic model), the smaller effective buffers will result in higher loss, which is of serious concern to network designers and operators who operate their router buffer sizes in this region.

The model presented above is highly simplified and ignores several aspects of TCP dynamics as well as real-time traffic characteristics. It nevertheless provides valuable insight into the anomaly, and will be used in later sections for a quantitative understanding of the impact of various parameters on the severity of the anomaly.

3.3. A MARKOV MODEL OF THE ANOMALY

It is in general challenging to mathematically analyse finite buffer systems in which several thousand feedback-based adaptive TCP flows interact with stochastic real-time traffic. We now develop a realistic yet rigorous Markov model based on some simplifications:

Assumption: *TCP packet arrivals are Poisson.* Figures 2 and 3 showed that if a large number (potentially thousands) of TCP flows multiplex at a bottleneck link with very small buffers, they do not synchronise their window dynamics behaviour, and can thus be treated as independent flows. Combined with the fact that each TCP flow's window will be quite small (since bottleneck buffers are small), implying that each flow will only generate a small amount of traffic per RTT, the aggregation of a large number of such independent flows can reasonably be assumed to yield Poisson traffic.

Assumption: *UDP packet arrivals are also Poisson.* Stochastic studies such as [14, 15, 24] have shown that the aggregation of traffic from a large number of independent flows (as can be expected at a core link) converges to Poisson. This important result makes the analysis tractable (though the phenomenon of anomalous loss is observed even if UDP arrivals are non-Poisson)

Claim: *UDP packets are on average smaller in size than TCP packets*, as discussed in Section 3.1., and as reported in several measurements of traffic in the Internet core [21]. Consistent with our example presented in Fig. 6, we choose average TCP and UDP packet sizes to be 1000 and 200 Bytes respectively.

Claim: *The aggregate TCP rate increases exponentially with bottleneck link buffer size*, as demonstrated in Fig. 7 and discussed in the previous section. Denoting the bottleneck buffer size as B (in KiloBytes), the TCP throughput λ_{TCP} is given by:

$$\lambda_{TCP} \approx \{1 - (e^{-B/B^*})\} \times \lambda_{TCP}^{sat} \tag{3}$$

where λ_{TCP}^{sat} denotes the saturation throughput of TCP (when buffer size is large).

In order to construct a Markov chain model we make the further assumption that packet transmission times are exponentially distributed (we will relax this assumption in the next subsection). We can then model the FIFO queue at the bottleneck link router as an M/M/1 system with finite buffer B and with two classes of customers:

1. UDP arrivals are Poisson at fixed rate (denoted by λ_{UDP}), and require exponential service time with unit mean (the service rate is normalised to average UDP packet size), and

2. TCP arrivals are Poisson at rate λ_{TCP} derived from Eq. (3), where each TCP packet arrival brings a bulk of 5 customers (corresponding to the packet size ratio $1000/200$), each requiring exponential service time with unit average.

For illustrative purposes, let us consider the buffer size B to be 3 KiloBytes. Then, we can model the state of the system as the number of customers in the FIFO queue. Fig. 9 shows the resulting Markov chain. A transition from state j to state $j+5$ corresponds to the arrival of a TCP packet, whereas a transition from state j to state $j+1$ corresponds to the arrival of a UDP packet.

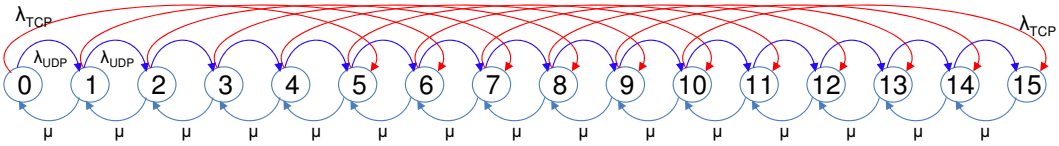


Figure 9. Markov chain state transition diagram for buffer occupancy with buffer size = 3000 Bytes

Denoting $B_{bytes} = B \times 1000 = 3000$ to be the corresponding buffer size in Bytes, and N the number of states in the Markov chain, then

$$N = \frac{B_{bytes}}{UDP\ packet\ size} + 1 = \frac{3000}{200} + 1 = 16. \tag{4}$$

If p_j represents the steady state probability of the queue being in state j (i.e., the probability that the queue contains j customers), then we can write the global balance equations as follows:

$$p_0(\lambda_{UDP} + \lambda_{TCP}) = p_1\mu \tag{5}$$

$$p_i(\lambda_{UDP} + \lambda_{TCP} + \mu) = p_{i-1}\lambda_{UDP} + p_{i+1}\mu \qquad (1 \le i \le 4) \tag{6}$$

$$p_i(\lambda_{UDP} + \lambda_{TCP} + \mu) = p_{i-1}\lambda_{UDP} + p_{i+1}\mu + p_{i-5}\lambda_{TCP} \qquad (5 \le i \le 10) \tag{7}$$

$$p_i(\lambda_{UDP} + \mu) = p_{i-1}\lambda_{UDP} + p_{i+1}\mu + p_{i-5}\lambda_{TCP} \qquad (11 \le i \le 14) \tag{8}$$

$$p_{15}\mu = p_{14}\lambda_{UDP} + p_{10}\lambda_{TCP} \tag{9}$$

The above equations and the normalising constraint $\sum_{i=0}^{15} p_i = 1$ form a set of linear equations that can be solved to compute the probability that an incoming UDP packet will be dropped, which in this example is p_{15}. Obtaining balance equations as the buffer size B increases is straightforward, and the resulting set of linear equations is easily solvable numerically (in MATLAB) to obtain the UDP packet loss probability.

The analytical result shown in this paper chooses model parameters to match the simulation setting as closely as possible: the normalised UDP rate is set to $\lambda_{UDP} = 0.05$ (i.e. 5% of link capacity), and the TCP saturation throughput $\lambda_{TCP}^{sat} = 0.94/5$ (so that TCP and UDP customers have a combined maximum rate less than the service rate of $\mu = 1$ in order to guarantee stability). The constant $B^* = 7$ KB is chosen consistent with what is obtained from simulations in Fig. 7.

Fig. 10 plots the UDP loss (on log scale) obtained from solving the M/M/1 chain with bulk arrivals and finite buffers, as well as the TCP rate in Eq. (3), as a function of buffer size B. It can be observed that in the region of 1-8 KB of buffering, UDP loss falls monotonically with buffer size. However, in the buffer size region between 9-30 KB, UDP packet loss increases with increasing buffer size, showing that the model is able to predict the anomaly found in simulations.

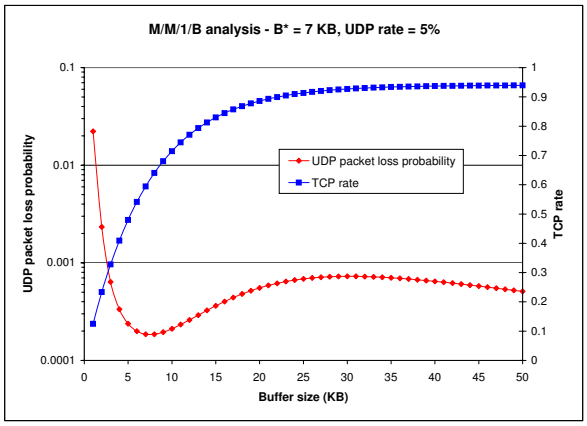


Figure 10. Anomalous UDP loss results from the M/M/1/B analytical model

3.4. SECTION SUMMARY

Prior work on buffer sizing focus purely on TCP performance. Although real-time (UDP) traffic accounts for only about 5-10% of Internet traffic, we note that its popularity, through the prolific use of on-line gaming, real-time video conferencing, and many other multimedia applications, is growing in the Internet. Consequently, we examined the dynamics of UDP and TCP interaction at a bottleneck link router equipped with very small buffers and observed a curious anomalous loss phenomenon. Further, we developed simple analytical models that gave insights into why the anomaly exists under certain circumstances.

We point the interested reader to our papers in [25–27] for a systematic exploration of the impact of various system parameters on the anomaly, and to [28] for a possible way of alleviating it.

4. REALISING SMALL BUFFER NETWORKS USING TRAFFIC PACING

The maturation of Wavelength Division Multiplexing (WDM) technology in recent years has made it possible to harness the enormous bandwidth potential of an optical fibre cost-effectively. As systems supporting hundreds of wavelengths per fibre with transmission rates of 10-40 Gbps per wavelength become available, electronic switching is increasingly challenged in scaling to match these transport capacities. However, due to the absence of a cost-effective technology for storing optical signals, emerging optical packet switched (OPS) networks are expected to have severely limited buffering capability.

A fundamental concern in OPS networks is packet loss due to contention, which occurs at a switching node whenever two or more packets try to leave on the same output link, on the same wavelength, at the same time. Many schemes have been developed to address contention losses - ranging from the use of fibre delay lines (FDLs) to store packets, deflection routing, and wavelength conversion. Unfortunately, these schemes have their own deficiencies: incorporating FDLs into a typical OPS switch design requires larger optical crossbars, which can add significantly to cost as the FDL buffers increase. Deflection

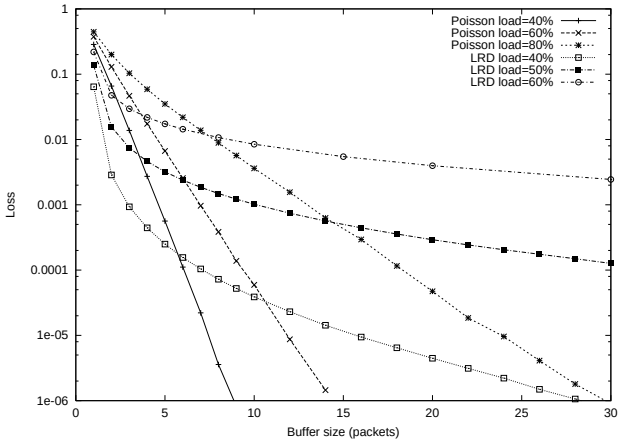


Figure 11. Loss vs. buffer size at a finite-buffer switch

routing usually incurs overheads such as packet reordering and implementation complexity, while all-optical wavelength converters are expensive, and often limited in their conversion range. It therefore seems that OPS networks of the foreseeable future will have very limited contention resolution resources.

4.1. SMALL BUFFERS: LOSS FOR REAL-TIME TRAFFIC

We first illustrate via simulation the impact of small buffers on losses for real-time traffic. A direct and obvious impact of small network buffers is an increase in packet losses. We begin by observing that not withstanding the high bandwidth available in OPS networks, small buffers at OPS nodes cause significant loss when the traffic exhibits short-time-scale burstiness. As an example we consider a single link with a queue of finite and small capacity. The link rate is set at 10 Gbps, and packets have a constant size of 1250 bytes (this is consistent with earlier studies of slotted OPS systems). Fig. 11 shows the packet losses as a function of buffer size obtained from simulations of short range (Poisson) as well as long range dependent (LRD) input traffic at various system loads. The plots illustrate that an OPS node with very limited buffering (say 10 to 20 packets) can experience significant losses even at low to moderate traffic loads, particularly with the LRD model which is more representative of real-world traffic. This may be unacceptable in a core network supporting real-time applications with stringent loss requirements.

Pacing, also known as smoothing, has been studied before in the context of video traffic transmission. Unlike a shaper, which releases traffic at a given rate, a pacer accepts arbitrary traffic with given delay constraints, and releases traffic that is *smoothest* subject to the time-constraints of the traffic. Here "smoothness" may be measured using the maximum rate, rate variance, etc. The delay tolerance of traffic passing through the pacer is crucial to the efficacy of the pacer – the longer the traffic can be held back at the pacer, the more the window of opportunity the pacer has to smooth traffic and reduce its burstiness.

To the best of our knowledge, there has not been a study on the use of traffic pacing techniques for alleviating contentions in OPS networks with very small buffering resources. Our packet pacer smoothes traffic entering the OPS network, and is therefore employed at

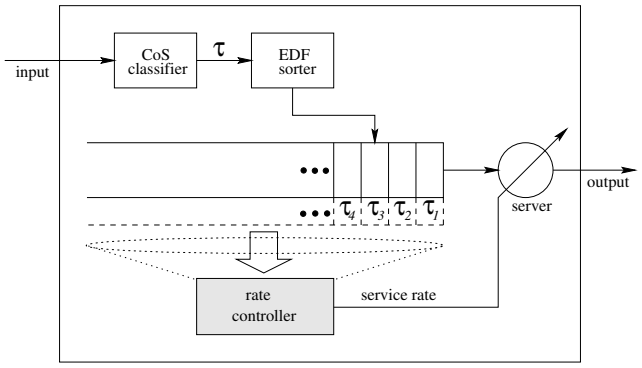


Figure 12. Model of the pacer

the optical edge switches on their egress links connecting to the all-optical packet switching core. Note that the optical edge switches process packets electronically, and are therefore assumed to have ample buffers required to do the pacing. Once a packet enters the OPS core, is it processed all-optically by each OPS core switch, where buffering is limited. The idea of pacing is therefore to modify the traffic profile entering the OPS network so as to use the limited buffers more efficiently and reduce losses, but without adversely affecting end-to-end delay.

4.2. PACING ENGINE AND OFF-LINE OPTIMUM

A generic architecture of our pacer is shown in Fig. 12. Incoming packets are classified (according to some criteria) and assigned a deadline by which they are to be released by the pacer. The objective of the pacer is to produce the smoothest output traffic such that each packet is released by its deadline. It is natural for the pacer therefore to release packets in order of deadline, namely to implement Earliest Deadline First (EDF) service.

Our pacing strategy derives from studies of video traffic smoothing, which we summarise next. Let $[0,T]$ denote the time interval during which the pacing system is considered, chosen such that the system is devoid of traffic at 0 and T. Denote by $A(t), 0 \leq t \leq T$ the *arrival curve*, namely the cumulative workload (say in units of bytes) arriving in $[0,t)$. Denote by $D(t), 0 \leq t \leq T$ the *deadline curve*, namely the cumulative workload that has to be served in $[0,t)$ so as not to violate any deadlines (thus any traffic with deadline earlier than t contributes to $D(t)$). Fig. 13 depicts an example $A(t)$ and $D(t)$ for the case where all arriving traffic has identical delay requirements. Note that by definition $D(t)$ can never lie above $A(t)$. Any service schedule implemented by the pacer can be represented by an *exit curve* $S(t), 0 \leq t \leq T$, corresponding to the cumulative traffic released by the pacer in $[0,t)$. A feasible exit curve, namely one which is causal and satisfies the delay constraint, must lie in the region bounded above by the arrival curve $A(t)$, and below by the deadline curve $D(t)$.

Amongst all feasible exit curves, the one which corresponds to the smoothest output traffic, measured by various metrics such as transmission rate variance, has been shown in [29] to be the *shortest path* between the origin $(0,0)$ and the point $(T,D(T))$, as shown in Fig. 13. This curve always comprises a sequence of straight-line segments joining points on

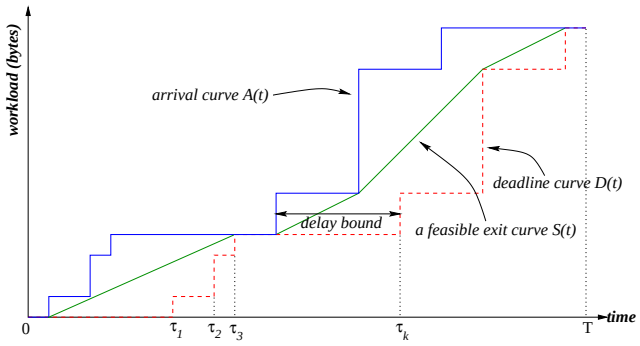


Figure 13. Arrival, deadline, and exit curves for an example workload process

the arrival and deadline curves, each segment representing a period during which the service rate is a constant. Computation of this curve requires knowledge of the complete traffic arrival curve, which restricts the approach to off-line applications like the transmission of stored video files.

Unlike the video transmission context, smoothing or pacing in OPS networks will have to operate under much more demanding conditions. Current mechanisms for smoothing consider one or a few video streams at end-hosts or video server; by contrast OPS edge nodes will have to perform the pacing on large traffic aggregates at extremely high data rates. The time-constraints for computing the optimal pacing patterns are also much more stringent – unlike video smoothing where a few frames (tens to hundreds of milliseconds) of delay is acceptable, OPS edge nodes will have to buffer traffic for shorter time lest the buffering requirement becomes prohibitively expensive (at 10 Gbps, 1 msec of buffering needs 10 Mbits of RAM). The next subsection therefore develops algorithms that are amenable to efficient implementation at OPS edge nodes.

4.3. EFFICIENT REAL-TIME PACING

It is shown in [29] that an off-line pacer yields the smoothest output traffic satisfying the delay constraints if its service rate follows the shortest path lying between the arrival and deadline curves. In the on-line case, however, the packet arrival process is non-deterministic, and the arrival curve is not known beforehand. In the absence of any assumptions about future packet arrivals, our on-line algorithm determines the smoothest output for the packets *currently* in the pacer. Thus at time t, the arrival curve considered to the right of t is a horizontal line (since future arrivals are not known yet), and the shortest-path exit curve degenerates to the convex hull of the deadline curve [30]. Upon each packet arrival, the deadline curve is augmented, and this may require a recomputation of the convex hull which defines the optimal exit curve. This section develops algorithms for the efficient update of the convex hull of the deadline curve upon each packet arrival.

4.3.1. SINGLE DELAY CLASS – CONSTANT AMORTISED COST ALGORITHM

We first consider the case where all packets entering the pacer have identical delay constraints. This simplifies the hull update algorithm since each packet arrival augments the

```
    // determine length and deadline of new packet p
1.  L = length(p); T = currtime; T_p = T + d
    // append new hull piece
2.  h = new hullPiece
3.  h.startT = ((hullList.empty()?) T : hullList.tail().endT);
4.  h.endT = T_p;
5.  h.slope = L/(T_p-h.startT)
6.  hullList.append(h)
    // scan backwards to restore hull convexity
7.  h = hullList.tail()
8.  while ((hPrev=h.prev)≠NULL ∧ hPrev.slope ≤ h.slope)
9.      h.slope = [h.slope ∗ (h.endT − h.startT)
                    + hPrev.slope ∗
                    (hPrev.endT − max(T, hPrev.startT))]
              / (h.endT − max(T, hPrev.startT))
10.     hullList.delete(hPrev)
11. end while // the hull is now convex
```

Figure 14. On-line algorithm for hull update upon packet arrival

deadline curve at the end. Our first algorithm computes the convex hull in $O(1)$ amortised time per packet arrival.

Fig. 14 depicts this update algorithm performed upon each packet arrival, and Fig. 15 illustrates the operations with an example. Recalling that the convex hull is piecewise-linear, we store it as a doubly linked list, where each element of the list corresponds to a linear segment whose start/end times and slope are maintained. In step 1 of the algorithm, the length of the incoming packet is determined, along with its deadline. The arrival of this new packet causes the deadline curve to be amended, which results in a new segment being appended to the hull. Steps 2-6 therefore create a new linear segment with the appropriate slope and append it to the end of the hull (shown by operation a in Fig. 15). The new piece may cause the hull to lose convexity, since the newly added piece may have slope larger than its preceding piece(s). Steps 7-11 therefore scan the hull backwards and restore convexity. If a hull piece has slope larger than its preceding piece, the two can be combined into a single piece which joins the end-points of the two pieces (as depicted by operations b and c in Fig. 15). The backward scan repeatedly fuses hull pieces until the slope of the last piece is smaller than the preceding piece (operation d in Fig. 15). At this stage the hull is convex and the backward scan can stop, resulting in the new hull.

4.3.2. GENERAL POLY-LOGARITHMIC COST ALGORITHM

We now consider the general case where arriving packets may have arbitrary delay constraints. Handling packets with different delay times is complicated by the fact that the arrival of a new packet causes significant changes to the deadline curve. Recalling that the deadline curve is a piecewise-linear curve, where the start/end times of its individual seg-

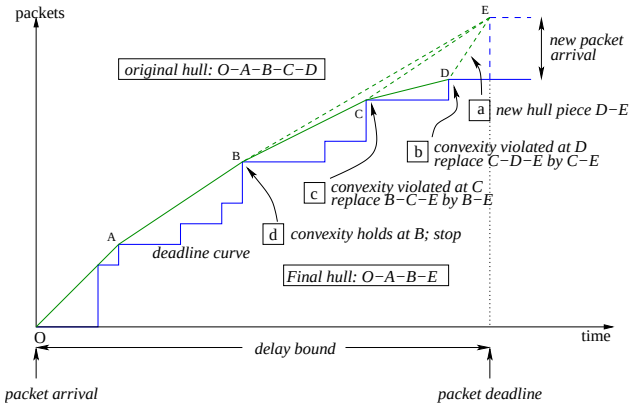


Figure 15. Example showing single-class hull update in amortised $O(1)$ time

ments correspond to deadlines of packets already in the system, we represent it as a planar polygonal line whose vertices $v_0, v_1, \ldots v_n$ form a sequence in increasing order with respect to both axes. The arrival of the new packet with a deadline between two existing vertices, say v_i and v_{i+1}, changes the deadline curve through the insertion of a new vertex u between them in the sequence and raising each of the vertices in the sub-sequence $v_{i+1}, v_{i+2}, \ldots, v_n$ by a value corresponding to the size of the new packet. As a result some vertices of the deadline curve, which were not part of the hull prior to arrival of the new packet, may appear as convex hull vertices as illustrated in Fig. 16. The number of such new points can be as high as the number of packets in the system, and the process of re-computing the convex hull is not as simple as searching a binary tree as we did in the single-class case above.

The idea behind the algorithm is that for an incoming packet with arbitrary deadline, the original deadline curve is split into two parts, corresponding to the left and right of the new arrival's deadline. The convex hulls for each of the parts is independently computed, after the deadline curve to the right has been shifted up to account for the new packet arrival. The two hulls are then merged back to get the complete convex hull. The goal is to perform this process as efficiently as possible.

Fig. 17 depicts our algorithm for determining the convex hull upon each packet arrival. Our vertices are stored in the leaves of a search tree T structure which is capable of supporting concatenable-queue operations with the value of time used as the search key. Each internal node of T stores the convex hull of its leaves in a secondary tree structure that is also capable of supporting concatenable-queue operations. A linear size of the tree T and all its secondary structures is achieved by storing a vertex in the convex hull of an internal node only if it is not stored in any of its ancestor nodes [31].

In step 1 of the algorithm, the size of the incoming packet is determined, along with its deadline, and a new vertex u is created. The arrival of the new packet, which causes the deadline curve to be altered, triggers re-computation of the convex hull. Step 2 therefore searches the tree T along the root-to-leaf path and inserts the new vertex u as a new leaf according to its deadline value. The tree T is then *divided* about u so that all the leaves to the left of u and u itself are in one 2-3 tree T_L and all the leaves to the right of u are in a second 2-3 tree T_R. The division is a recursive process. For each internal node visited during

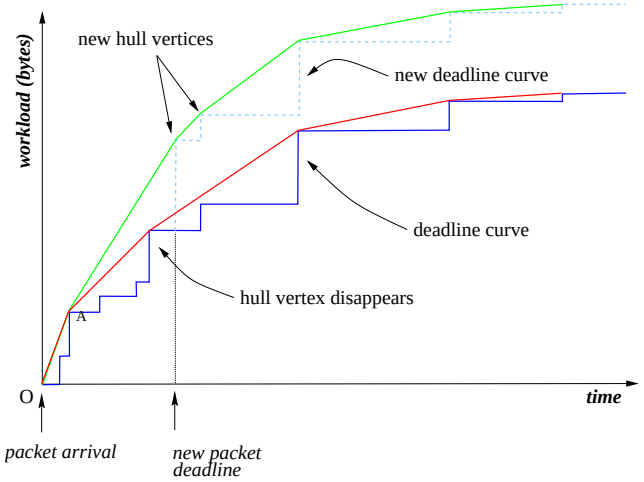


Figure 16. Example illustrating that an arbitrary number of new hull points may appear when the arriving packet has an arbitrary deadline

1. *Determine size and deadline of newly arrived packet and create new vertex u.*
2. *Insert u into the 2-3 tree and divide it into trees T_L and T_R. T_L holds keys $\leq$ that of u, and T_R the remaining. Store size of new packet in root of T_R.*
3. *Merge T_L and T_R into a single tree.*

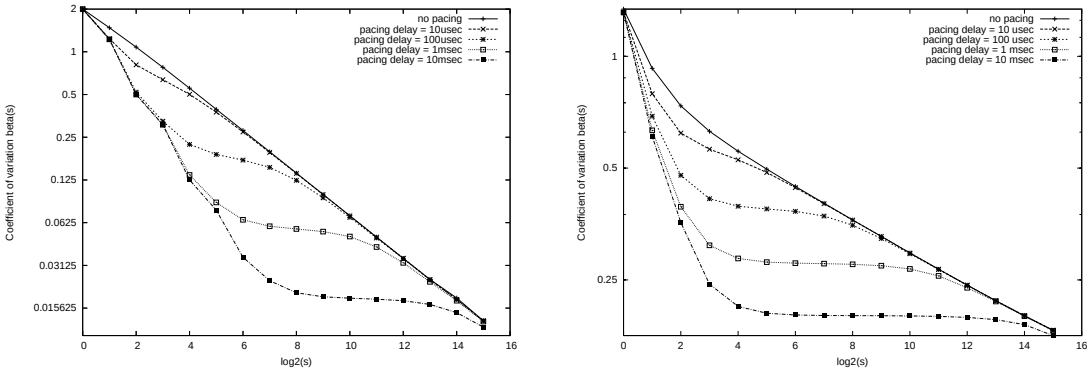
Figure 17. Multi-class hull update algorithm

the search process that will be deleted in the divide process, we use the convex hull stored in its secondary structure to compute a complete hull for each of its children, as described in [31]. At the end of the divide process, values of all vertices in T_R need to be incremented by the size of the incoming packet (i.e. shift the deadline curve up); this is achieved by storing the size of the new packet in the root of T_R. In step 3, the two trees T_L and T_R are again concatenated into a single tree, which yields the final convex hull of the new deadline curve. The complexity of the entire convex hull update operation that need to be performed upon each packet arrival is $O(\log^2 n)$, where n is the number of queued packets [31].

As in the single-class case, a hull segment needs to be deleted once packets corresponding to the segment have been released. The complexity of deleting a vertex of the convex hull and restructuring the tree T again has an $O(\log^2 n)$ cost, where n is the number of queued packets, as shown in [31].

4.4. PERFORMANCE EVALUATION FOR A SINGLE FLOW

Having addressed the *feasibility* of pacing at high data rates, we demonstrate the *utility* of pacing in OPS systems with small buffers. This section evaluates via simulation the impact of pacing on traffic burstiness and loss for a single flow, while the next section evaluates via simulation loss performance for several flows in realistic network topologies.



(a) Poisson traffic (b) LRD traffic

Figure 18. Burstiness vs. time-scale for various pacing delays from simulation of Poisson and LRD traffic

We apply our pacing technique to Poisson and long range dependent (LRD) traffic models (both of which were introduced in section 4.1.); the Poisson model is selected for its simplicity and ease of illustration of the central ideas, while the LRD model is chosen since it is believed to be more reflective of traffic in real networks.

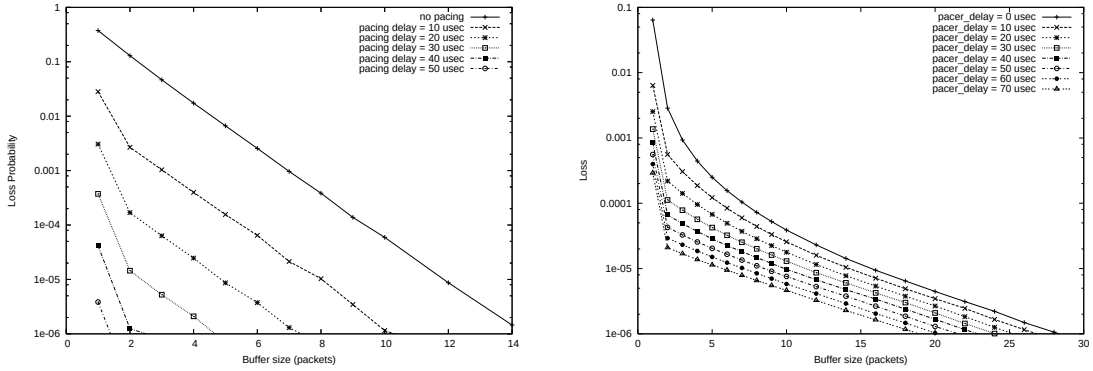
4.4.1. IMPACT OF PACING ON TRAFFIC BURSTINESS

We now study using simulation and analysis how pacing changes the burstiness of a traffic stream at various time-scales. Burstiness at time-sale s is quantified by $\beta(s)$, the coefficient of variation (i.e. ratio of standard deviation to mean) of traffic volume measured over time intervals of size s. Log-log plots of $\beta(s)$ versus s are routinely used in the literature to depict traffic burstiness over various time-scales as an indicator of self-similarity of traffic traces and to show the influence of the Hurst parameter H. Our simulations in this section fix the link rate at 10 Gbps and packet sizes at 1250 bytes (such that each packet requires exactly 1μs for transmission).

Fig. 18 shows for Poisson and LRD traffic the burstiness $\beta(s)$ versus time-scale s (in μsec) on log-log scale observed in simulation for pacing delay d of 0 (i.e. no pacing), 10μsec, 100μsec, 1msec, and 10msec. We first note that the unpaced Poisson stream (at 8 Gbps or 80% load) in Fig. 18(a) and the unpaced LRD stream (at 4 Gbps or 40% load) both exhibit straight lines with respective slopes -0.5 and -0.15; this validates the expected slope $-(1-H)$ for Hurst parameter settings $H = 0.5$ and $H = 0.85$ of the short and long range dependent traffic respectively (the different slope at very short time-scales for LRD traffic arises from the packetisation of the ideal fluid model).

We next note that pacing reduces burstiness only within a range of time-scales, explained as follows. At ***very short*** time-scales (e.g. $s = 2^0 = 1\mu$sec in our example), burstiness is invariant to pacing due to the discrete nature of packet release. Specifically, if $p_i(\Delta t)$ denotes the probability that an interval of size Δt has i packets, then for $i \geq 2$: $\lim_{\Delta t \to 0} p_i(\Delta t) = o(\Delta t)$ is vanishingly small when the traffic model does not permit batch arrivals. The burstiness $\beta(\Delta t)$ at very short time-scales therefore depends only on $p_0(\Delta t)$

and $p_1(\Delta t)$, which in turn are determined by the mean traffic rate and are invariant to pacing. At ***very long*** time-scales (beyond the delay budget d of the pacer), burstiness is again invariant to pacing. This is because the pacer does not hold any packet back beyond its deadline, and so pacing cannot alter the characteristics of the traffic at time scales that are much longer than the pacing delay budget.



(a) Poisson traffic (b) LRD traffic

Figure 19. Loss versus buffer size at various pacing delays from simulation of Poisson and LRD traffic

Pacing is most effective in the range of time-scales within the pacer's delay budget. Barring the very-short time-scale region (that is dominated by the discrete nature of the packet pacing process), we observe that for both Poisson and LRD traffic, and for any fixed pacer delay budget d, the burstiness of the paced traffic remains nearly constant at $\beta(d)$ with time-scale, till it converges with the burstiness of the input traffic. This demonstrates the efficacy of pacing in reducing short time-scale burstiness, without altering longer time-scale traffic characteristics.

4.4.2. IMPACT OF PACING ON PACKET LOSS

Having quantified the impact of pacing on the burstiness of Poisson and LRD traffic, we now feed the paced traffic into a constant rate server (OPS link) and observe how losses are affected by pacing. For analytical and conceptual simplicity we consider only a single flow with fixed size packets in this section; realistic network topologies with multiple flows and variable packet sizes are considered in the subsequent section.

In our simulation we feed the paced traffic stream into an OPS link with fixed capacity and small buffers, and observe the packet loss probability as a function of buffer size for various pacing delays. Fig. 19(a) plots for Poisson traffic (at 60% link load) the observed loss (on log scale) as a function of buffer size (in packets) for pacing delays of 0 (corresponding to no pacing), 10, 20, 30, 40, and 50 μsec (recall that 1μsec corresponds to the transmission time of a packet). Pacing is seen to be extremely effective in reducing loss: at the cost of a few tens of μsec of increase in end-to-end delay, the packet loss probability can be reduced by multiple orders of magnitude, which is a very attractive cost-benefit trade-off. Fig. 19(b) plots the losses for an LRD traffic stream (at 40% load) when pacing delay of 0 to 70μsec is employed. Once again pacing is seen to be effective: for example, a 70μsec pacing delay

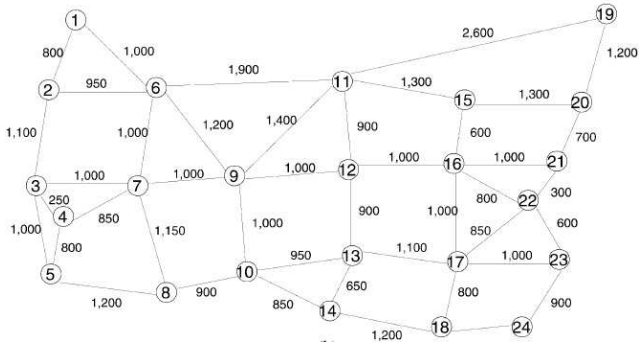


Figure 20. Core Network Simulation Topology

(which contributes to a very small increase in end-to-end delay), reduces losses at the link by more than an order of magnitude.

4.4.3. SIMULATION STUDY OF NETWORKS

Our second set of experiments simulate the core network topology shown in Fig. 20 comprising of 24 core OPS nodes. Each core node is connected to four edge nodes (not shown in the figure), and the network thus has 96 edge nodes. Each edge node generates traffic to one other randomly chosen edge node, giving us a total of 96 flows in the network. Each flow is routed along the shortest path (computed using the shown fibre lengths) from origin to destination. All flows generate Poisson traffic (we did not have sufficiently long traces of LRD traffic to generate sufficient packets for this large topology), and the maximum core link load in the network is maintained between 64% and 80% of link capacity. In the next subsection, we discuss another scheme to alleviate contention losses in a bufferless core network.

Fig. 21 shows the aggregate packet loss probability in the network as a function of buffer size (in bytes) when optical buffering is employed for contention resolution. Each curve in the figure corresponds to a different pacing delay. Once again pacing is seen to be very effective in reducing core loss for a very small penalty in end-to-end delay: for 10-20 packets (4200-8400 bytes) of buffering, a pacing delay of 100μs reduced loss by as much as three orders of magnitude, which would be a very attractive loss-delay trade-off in a core OPS network.

For in-depth analytical and simulation results, we encourage the reader to refer our papers in [32, 33].

4.5. SECTION SUMMARY

Emerging optical packet switched (OPS) networks will likely have very limited contention resolution resources usually implemented in the form of packet buffers or wavelength converters. This can cause high packet losses and adversely impact end-to-end performance. We identify short-time-scale burstiness as the major contributor to the performance degradation, and proposed to mitigate the problem by "pacing" traffic at the optical edge prior to injection into the OPS core. Pacing dramatically reduces traffic burstiness for a bounded

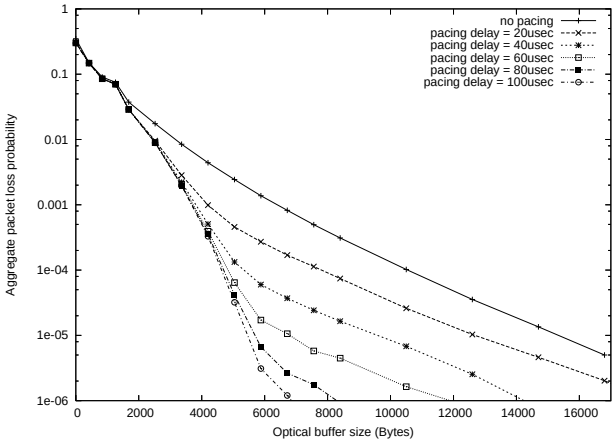


Figure 21. Loss versus buffer size

and controllable penalty in end-to-end delay. We presented algorithms of poly-logarithmic complexity that can efficiently implement optimal real-time pacing of traffic aggregates with arbitrary delay requirements. We showed via simulation of realistic OPS network topologies that pacing can reduce losses by orders of magnitude, at the expense of a small and bounded increase in end-to-end delay. This ability to trade-off delay for loss makes pacing a very attractive way of realising acceptable performance from OPS networks with small buffers.

5. REALISING BUFFERLESS NETWORKS USING PACKET-LEVEL FEC

From the discussions in the previous sections it is clear that as router line card rates continue to increase, we are approaching the limits of semiconductor (SRAM and DRAM) technology in terms of switching speeds, power savings and heat dissipation. This observation has forced high capacity router/switch designers, and network providers to consider leveraging the use of optics for switching and transmission in core routers.

However, incorporating even a few packets of buffering using an all-optical on-chip memory is a formidable challenge due to the inherent complexity associated with maintaining the quality of the optical signal and the physical size limitations of the chip [34]. At the moment, our IRIS router is capable of buffering 100 nanosec worth of data. At 40 Gbps line rate, this translates to 500 Bytes, which is not sufficient to buffer a typical 1500 Byte Internet packet.

Given these challenges and limitations associated with all-optical buffering, we investigate if we can enable a high-speed wide-area bufferless (or near-zero buffer) core optical network capable of delivering acceptable end-to-end performance. The concern in a bufferless network is that packet losses can be unacceptably high. We propose the use of packet-level forward error correction (FEC) coding by the electronic edge routers as a method for recovering from packet loss in the bufferless core.

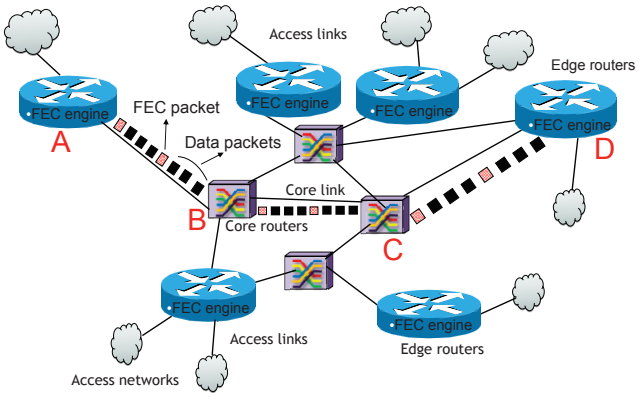


Figure 22. Topology to illustrate the edge-to-edge FEC framework

5.1. MOTIVATION FOR CHOOSING FEC

An interesting question that we ask in the context of a bufferless core network is: do we expect that the losses in the core will be bursty? Losses in packet networks are known to be bursty [16], but that is in the case of drop-tail queues and also when the lossy links are the flows' bottlenecks. In practice, the capacity of the core links is orders of magnitude higher than the access/edge links [8], thus ensuring that the bandwidth bottleneck for a flow is either at the access/edge link, and not at any core link. Thus, losses will occur in the core, not because a single flow can saturate the core link with a burst of back-to-back packets, but because we can have the simultaneous arrival of two or more packets from different edge/core links in a given time-slot. Thus, loss at core links is due to *contention*, not congestion.

It is well-known that FEC works best when losses are random, and hence our choice of using FEC is primarily inspired by this fact that packet losses in a bufferless core will occur randomly (due to contention) and not in a bursty manner (due to congestion).

Other reasons for choosing FEC are that it is a well established technique, cost-effective, and can be easily implemented in hardware. FEC can introduce some bandwidth overhead, but this is a small price to pay for building scalable and power efficient bufferless core nodes, and also because ISPs typically operate their core networks at relatively low loads ($\approx$ 20-30%) [35]. In addition, packet-level FEC has, to the best of our knowledge, not been studied before in the context of a bufferless network. Finally, it also complements other techniques outlined above to minimise loss, namely that it is possible to combine FEC with traffic shaping/pacing etc.

5.2. THE EDGE-TO-EDGE PACKET-LEVEL FEC FRAMEWORK

Fig. 22 shows a small segment of a typical ISP network comprising of electronic edge routers and optical core routers. The distinguishing feature between the core and edge routers is that the core router links are bufferless (near-zero buffer) while the electronic router links have large buffers. The FEC framework presented in this paper is implemented at the electronic edge routers across the aggregate packet flow between an edge router

(ingress) to another edge router (egress), and not for flows between any two end-hosts. Hence our implementation is an *edge-to-edge* based FEC implementation. As an example, all traffic that enters the core network from an ingress router say in New York city and exits at an egress router say in Los Angeles is viewed as an edge-to-edge flow, and it is protected by the FEC redundancy. It is important to note that the FEC scheme is scalable since if an ISP network has N edge routers, then each electronic edge router will compute FEC packets for just $N-1$ edge-to-edge flows. Further, the proposed FEC framework has the advantage of being completely transparent to the end-hosts with control purely with the ISP.

The ingress edge routers receive traffic from applications running at various end-hosts on the access network via access links (DSL, cable modem, etc.), classifies the traffic on an edge-to-edge basis, and computes the FEC packets per egress edge router. The FEC framework discussed in this work uses the well-known and simple XOR scheme. The strength of FEC is the number of data packets over which the XOR operation is performed, henceforth referred to as block-size. The scheme has a unique property, namely that if in addition to a block of k data packets the ingress router also transmits the XOR of the k data packets (thereby transmitting $k+1$ packets for every k packets), the corresponding egress router will be able to recover from loss provided there is only one missing data packet.

For ease of illustration, the figure shows traffic flowing from edge router A to edge router D along the path A-B-C-D. Assuming the block-size is three, A keeps a running XOR of the data packets destined to D. After every third packet, the FEC packet comprising the XOR value is transmitted and the XOR is cleared. D also maintains a running XOR of the packets it receives from A. If it detects exactly one lost data packet in the window of $k+1$ packets, the running XOR is XOR'd with the FEC packet to recover the lost data packet, which is then forwarded on. In the case of zero (or > 1) loss, recovery is not possible and the running XOR is cleared.

To deal with variable-size packets, we assume that the size of each FEC packet equals the MTU size in the optical core. For XOR purposes, smaller packets are treated as being padded with zeros. The egress router must have sufficient information to recover a missing data packet correctly. Therefore, the FEC packet constructed by the ingress router takes into account the header information and the payload of each data packet. This ensures that the reconstructed packet will be identical to the original (missing) data packet.

The edge routers do not introduce much overhead in computing the FEC packet (since the XOR computation can be performed in real-time), nor do they require significant additional memory for the XOR process. The extra memory required is for storing one FEC packet per edge router in the network. This is however not a concern because FEC is performed at the edge routers that have large electronic memory. Insertion of one FEC packet for every k data packets increases the bandwidth requirement by a fraction $1/k$, which may be acceptable in a typical optical core that has abundant bandwidth but limited buffering.

The recovery operation can introduce a delay that in the worse case is the time to receive all k subsequent packets in a window of $k+1$ packets (assuming the first data packet is lost). The delay will be very small in practice because we have eliminated the large buffers that exist in today's core routers (millions of packets), and we consider edge-to-edge flows with possibly thousands/millions of packets per second.

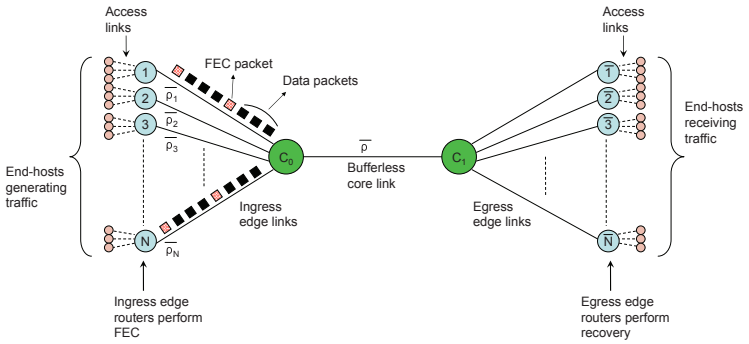


Figure 23. Example dumbbell topology with a single core link

5.3. ROLE OF FEC ON SINGLE LINK GOODPUT AND LOSS

We implemented the above edge-to-edge FEC framework in *ns-2* (version 2.33), and apply it to the single core-link dumbbell topology shown in Fig. 23. Ten edge links feed traffic into the bufferless core link at router C_0, with each edge link in turn fed by three access links. The 30 end-hosts each have 5 TCP (Reno) agents, and the network therefore simulates 150 long-lived TCP flows. Similarly the TCP flows are sinked by the 30 end-hosts on the right. The propagation delays on the access and edge links are uniformly distributed between $[1,5]$ ms and $[5,15]$ ms respectively, while the core link C_0-C_1 has delay 30 ms. In line with the prevalent situation in today's networks, we ensure that the TCP flows are bottlenecked at access links, rather than the core which typically has much higher capacity. Our access link speeds are uniformly distributed in $[3,5]$ Mbps, all edge links operate at 40 Mbps, and the core link at 400 Mbps. For these link speeds, it can be seen that the access link is the bottleneck since each flow's fair-share of the bandwidth on the access links varies between 0.6-1 Mbps, while on the edges and the core it is 2.67 Mbps. The start time of the TCP flows is uniformly distributed in the interval $[0,10]$ sec and the simulation is run for 35 sec. Data in the interval $[20,35]$ sec is used in all our computations so as to capture the steady-state behaviour of the network.

We measure the average per-flow TCP goodput for each setting of the FEC block-size k in simulation. We use goodput as a metric since it has been argued to be the most important measure for end-users [36], who want their transactions to complete as fast as possible. It should be mentioned that *ns-2* does not permit setting buffer size to zero as it simulates store-and-forward rather than cut-through switches. Thus, our simulations use a buffer size of 1 KB (the closest we can come to zero buffer) to accommodate a single TCP packet (TCP packets in our simulation are of size 1 KB) that is stored and forwarded by the switch.

Fig. 24 shows the per-flow TCP goodput as a function of block-size k. For comparison, it also depicts, via horizontal lines, the average goodput without FEC (the bottom line) and the average goodput if the core link were to have sufficient (delay-bandwidth) buffering of around 12.5 MB (top line). Large buffers yield a per-flow goodput of 0.7 Mbps, while eliminating buffers reduces this goodput to 0.5 Mbps, a sacrifice in goodput of nearly 30%. Employing edge-to-edge FEC over the bufferless link can improve per-flow goodput substantially, peaking at nearly 0.68 Mbps when the FEC block-size k is in the range of 3-6, and bringing the per-flow TCP goodput for the bufferless link to within 3% of a fully-buffered

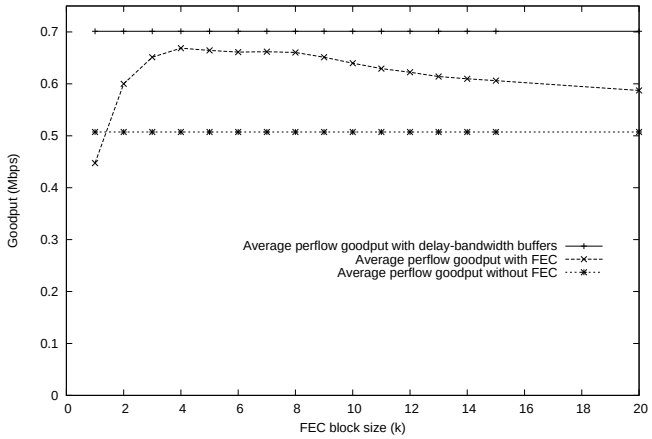


Figure 24. Average perflow goodput for 150 TCP flows on dumbbell topology

link. This small sacrifice in goodput is a worthy price to pay for eliminating buffering at router C_0.

Another interesting aspect to note from Fig. 24 is that TCP goodput initially increases with FEC block-size k, reaches a peak, and then falls as k increases. Qualitatively, this is because stronger FEC (i.e., smaller block-size k) in general improves the ability to recover from loss, but is also a contributor to loss since it increases the load on the link by introducing redundant packets. In the next subsection, we capture this effect via a simple analytical model to determine the optimal setting of FEC block-size that minimises loss on a bufferless link.

5.3.1. ANALYSIS

We develop a simple analytical model to quantitatively understand the impact of FEC strength on edge-to-edge loss, and to identify the block-size settings that achieve low loss and consequently larger goodput. Our analysis makes several simplifying assumptions:

1) The end-hosts that generate traffic are independent of each other. Consequently, traffic coming into the core links from the various edge links are also independent of one another. This is a reasonable assumption, even for TCP traffic when the number of flows is large enough [3]. Moreover, we assume that the contribution to the load on the core link from each of the edge links is similar.

2) We assume a time-slotted cut-through link, with loss happening if and only if two or more packets arrive to the link for transmission in the same slot (since the core is bufferless).

3) We do not model feedback, i.e., TCP's adjustment of rate in response to loss. Instead, we will assume that the steady-state load on the link is known aforehand.

Denote by ρ_i the original load (i.e., load without FEC) on each of the edge links (i, C_0), $i \in \{1, N\}$ (see Fig. 23). The offered load at the core link C_0-C_1 is then $\rho = \sum_{i=1}^{N} \rho_i$. Now, if each edge link performs FEC using block-size k, then the new load $\overline{\rho_i}$ on each of these edge links is

$$\overline{\rho_i} = \left(\frac{k+1}{k}\right)\rho_i \tag{10}$$

since FEC inserts one additional packet for every k data packets. Correspondingly, the offered load $\overline{\rho}$ post-FEC at the core link is

$$\overline{\rho} = \sum_{i=1}^{N} \overline{\rho_i} = \sum_{i=1}^{N} \left(\frac{k+1}{k}\right) \rho_i = \left(\frac{k+1}{k}\right) \rho \tag{11}$$

Assuming that each edge link contributes equally to the load $\overline{\rho}$ on the core link, the probability that in a given time-slot a packet arrives from any chosen edge link is $\overline{\rho}/N$ where N denotes the number of edge links. The loss probability $\mathbb{L}_c$ at the core link in a chosen slot is then the probability that packets arrive from two or more edge links:

$$\mathbb{L}_c = \sum_{i=2}^{N} \binom{N}{i} \left(\frac{\overline{\rho}}{N}\right)^i \left(1 - \frac{\overline{\rho}}{N}\right)^{N-i} \tag{12}$$

It is worthwhile mentioning that for large N, the binomial distribution above converges to the Poisson distribution with the same mean. The number of input links (N) interfacing with a core router is called fan-in, and is fairly large in practice. For example, the mean number of working ports in a core network with 60 nodes and uniform full mesh of demands is about 354 [37]. Therefore, Eq. (12) can be approximated by

$$\mathbb{L}_c = 1 - e^{-\overline{\rho}}(1 + \overline{\rho}) \tag{13}$$

Knowing the probability of packet loss in the core, we can now estimate the edge-to-edge packet loss probability $\mathbb{L}_e$ by computing the expected number of irrecoverably lost packets in a window of $k+1$ packets (comprising k data packets and one FEC packet) as follows:

$$\mathbb{L}_e = \mathbb{L}_c \sum_{j=1}^{k} \binom{k}{j} (\mathbb{L}_c)^j (1 - \mathbb{L}_c)^{k-j} \frac{j}{k} + (1 - \mathbb{L}_c) \sum_{j=2}^{k} \binom{k}{j} (\mathbb{L}_c)^j (1 - \mathbb{L}_c)^{k-j} \frac{j}{k} \tag{14}$$

The first term on the right in Eq. (14) captures the case when the FEC packet is lost along with j data packets, in which all j data packets are irrecoverable, while the second term captures the case when the FEC packet arrives and $j \geq 2$ data packets are lost, in which case the j packet losses are irrecoverable. Eq. (14) can be simplified yielding

$$\mathbb{L}_e = \mathbb{L}_c \left[1 - (1 - \mathbb{L}_c)^k\right] \tag{15}$$

Eq. (15) states that a data packet is irrecoverably lost only if it is lost in the core (with probability $\mathbb{L}_c$) **and** not all other k packets in the window (this includes the FEC packet) are successfully received (otherwise the lost data packet can be reconstructed).

Eq. (15), in conjunction with Eq. (12) (or Eq. (13)) and Eq. (11), can be used to directly estimate edge-to-edge loss $\mathbb{L}_e$ as a function of FEC block-size k. In Fig. 25 we plot on log-scale the edge-to-edge packet loss probability as a function of the block-size k for different values of load ρ $(20\%, 30\%, 40\%)$ and fan-in N. We first observe that for a given load, loss is not very sensitive to the fan-in N at the core link, and further that the

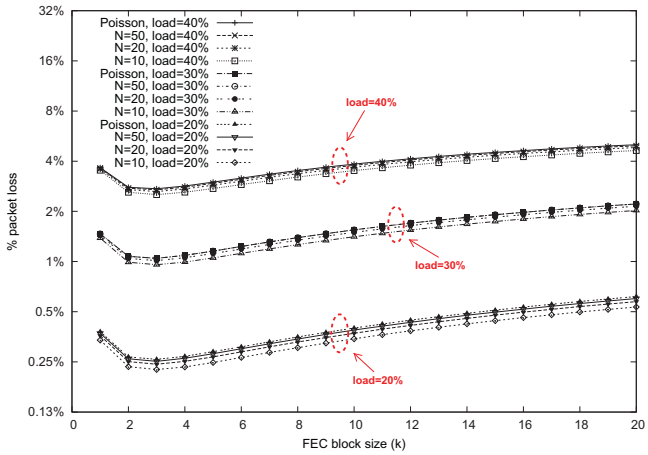


Figure 25. % edge-to-edge packet loss for different loads and fan-in

Poisson limit seems to be a good approximation that frees us from having to consider the fan-in parameter N explicitly in the model. The most important observation to emerge from this plot is that for a given load, the loss decreases with block-size k, reaches a minimum, and then starts increasing as the block-size gets larger. This provides some explanation as to why the simulation plot in Fig. 24 shows TCP goodput to first increase and then fall with block-size k, as TCP throughput is inversely related to the square root of end-to-end packet loss [38]. The figure also gives us some estimate of the strength of FEC required ($k = 3$ in this case) for minimising loss: the recovery benefit of stronger FEC (i.e., lower k) is outweighed by the overhead it introduces in terms of load, while weaker FEC (i.e., larger k) does not sufficiently recover lost data packets. In the next subsection we explore if similar observations extend to a more complex multi-hop topology.

5.3.2. FEC PERFORMANCE IN A MULTI-HOP NETWORK

Having seen the benefits offered by FEC for a single link, we now evaluate its performance on a more general wide-area network topology. To this end, we choose the NSFNet topology shown in Fig. 26 as our representative core network, which is made up of core routers (numbered 0 to 13) and the bufferless optical links interconnecting them. The numbers along the core links indicate the propagation delay in milliseconds. For the sake of clarity, the figure shows only two edge routers connected to every core node and each edge router receives traffic from only two end-hosts (via access links). However, in all our simulations, we consider larger number of edge/access links, as described next.

We consider ten edge links feeding traffic into every core router, and each edge router in turn is fed by five access links. All core links operate at 1 Gbps, all edge links at 100 Mbps, and the access link rates are uniformly distributed between $[7, 10]$ Mbps, to reflect a typical home user. These numbers ensure that the core is not the bottleneck for any TCP flow. The destination end-hosts are chosen randomly such that every flow traverses at least one hop on the core network; in all there are 3480 TCP flows in the network comprising of 784 one-hop flows, 1376 two-hop flows and 1320 three-hop flows. We assume all flows to be long-lived (Section V describes results when both short-lived and long-lived TCP flows

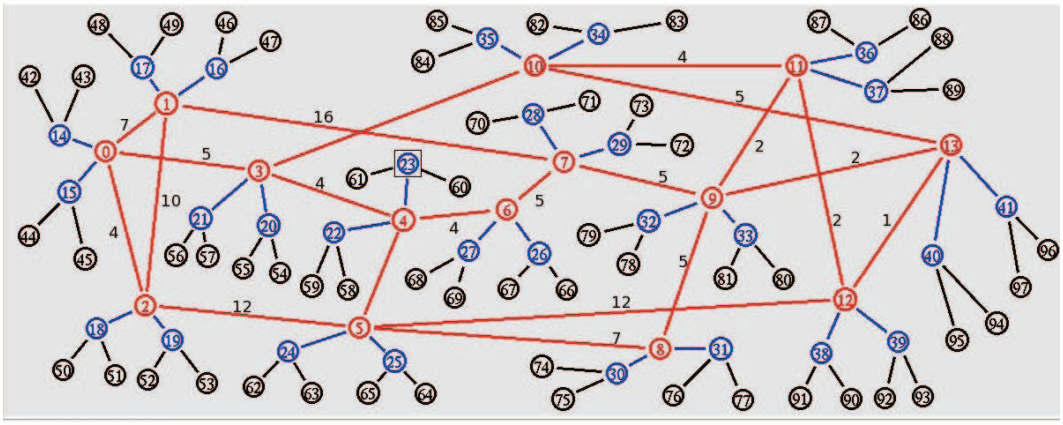


Figure 26. Example NSFNet topology with 2 access links per edge node and 2 edge links per core node from *ns-2* network animator

coexist). Data in the interval $[20,35]$ sec is used for the computations in order to capture the network's steady state behaviour.

Fig. 27 plots the ratio (average goodput with FEC to the corresponding average goodput with delay-bandwidth buffers) for 1-, 2- and 3-hop TCP flows as a function of the block-size (the maximum number of hops along the shortest path between any two core nodes on the NSFNet is three). It makes sense to use the goodput obtained with delay-bandwidth buffers in the core as the benchmark because core routers today have large buffers [1], and the performance witnessed by ISPs is typically under such large buffering. We note from the simulation results that with delay-bandwidth buffers, the load on the core links varies between 7% to 38% with the average load being $\approx 24\%$. These numbers are realistic and fall in the regime in which most ISPs operate their networks today. The figure also indicates, via horizontal lines, the corresponding goodput ratios in the non-FEC case.

A fundamental point that we can infer from the figure is that the bufferless core network (with and without FEC) is very unfair towards multi-hop flows when compared to single-hop flows. On average, 1-hop flows with FEC (at $k = 3$) achieve nearly 1.5 times the goodput (1.05 times in the non-FEC case) when compared to what they achieve when all core routers have delay-bandwidth buffers. This means that 1-hop flows perform better in a bufferless network than in a fully-buffered network! The ratio reduces to 0.56 for 2-hop flows and further to just 0.3 for 3-hop flows, indicating heavy skewing in favour of short-hop flows.

To explain why multi-hop TCP flows perform so poorly, we plot in Fig. 28 the histogram of edge-to-edge packet loss (for the non-FEC case) for flows with different hop-lengths. We can observe that while over 95% of 1-hop flows experience loss only in the range 0.5-3%, it increases to 1.5-4.5% for 2-hop flows, and further to 2.5-6% for 3-hop flows. To appreciate the impact these numbers have on the edge-to-edge performance, if we assume, for example, that the loss rate doubles from 1% to 2%, then the throughput of open-loop UDP traffic reduces by roughly 1%, whereas for closed-loop TCP traffic, it reduces by nearly 30%, since the average throughput of a TCP flow in the congestion avoidance mode is inversely proportional to the square root of packet loss. TCP goodput, however,

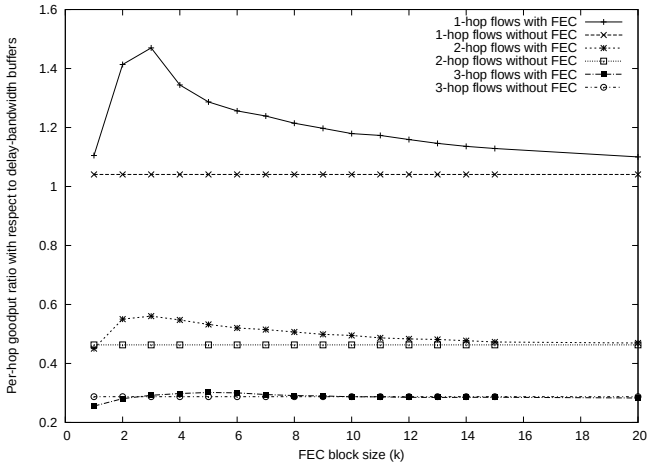


Figure 27. Ratio (average goodput with FEC to average goodput with delay-bandwidth buffers) for 1-, 2-, 3-hop TCP flows on NSFNet topology

can be much lower, as seen by Fig. 27. The relatively higher loss rates for 2- and 3-hop flows result in their fair-share of the bandwidth being unfairly utilised by 1-hop flows (since TCP is inherently greedy and is designed to exploit as much of the bandwidth as available), leading to unfairness. These results motivate us to devise a scheme that provides fairness to both single- and multi-hop flows, which will be the focus of the next section.

5.4. FEC FOR A BUFFERLESS NETWORK AND FAIRNESS

We observed from the results in the previous section that in a bufferless network, multi-hop TCP flows can experience significantly lower end-to-end goodput than single-hop flows, leading to unfairness. In this section, we address this deficiency by developing a framework that ensures fairness to both single- and multi-hop flows.

5.4.1. ANALYSIS

We denote $\lambda_{i,j}$ to be the offered load to the core network by the edge-to-edge flow between ingress router i and egress router j, henceforth represented as (i, j), under the assumption that they are not using FEC. Let $\overline{\lambda_{i,j}}$ be the new load to the core network when the flow employs FEC using $k_{i,j}$ as its block-size. Consequently,

$$\overline{\lambda_{i,j}} = \left(\frac{k_{i,j}+1}{k_{i,j}}\right)\lambda_{i,j} \tag{16}$$

Under the assumption that packet arrivals at a core link (u, v) in every time-slot is Poisson with mean $\overline{\lambda_{i,j}}$, we can compute the packet loss probability $L_c^{u,v}$ at the link as the probability

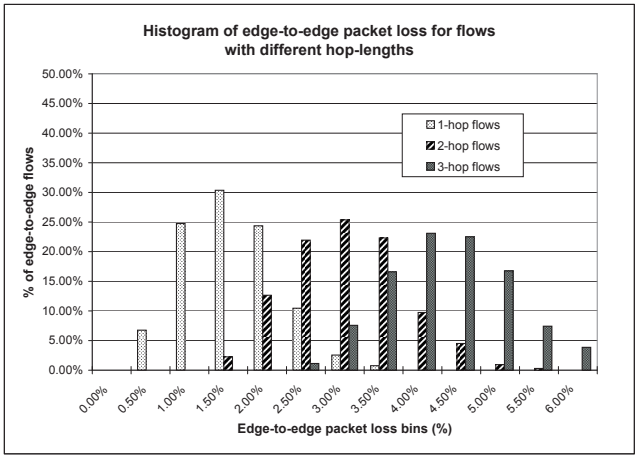


Figure 28. Histogram of edge-to-edge packet loss for TCP flows with different hop lengths (and no FEC) on the NSFNet

of two or more simultaneous packet arrivals from all flows that traverse (u,v). Thus,

$$L_c^{u,v} = 1 - \left(e^{-\sum_{(u,v)\in r(i,j)} \overline{\lambda_{i,j}}} \right) - \left(\sum_{(u,v)\in r(i,j)} \overline{\lambda_{i,j}} \times e^{-\sum_{(u,v)\in r(i,j)} \overline{\lambda_{i,j}}} \right)$$
$$= 1 - e^{-\sum_{(u,v)\in r(i,j)} \overline{\lambda_{i,j}}} \left(1 + \sum_{(u,v)\in r(i,j)} \overline{\lambda_{i,j}} \right) \quad (17)$$

where $r(i,j)$ is the routing path of edge-to-edge flow (i,j). In general, a flow can traverse multiple hops on the core network before reaching the egress edge router. If loss rates on core links are sufficiently small (say 10^{-2} or lower), it is reasonable to assume that edge-to-edge losses are independent and additive over the links the flow traverses. Therefore, denoting $\mathbb{L}_c^{i,j}$ to be the aggregate core path loss probability for the flow (i,j),

$$\mathbb{L}_c^{i,j} = \sum_{(u,v)\in r(i,j)} L_c^{u,v} \quad (18)$$

We can now compute $\mathbb{L}_e^{i,j}$, the edge-to-edge packet loss probability for flow (i,j) by substituting in Eq. (15) the core path loss probability for the flow derived from Eq. (18). Thus,

$$\mathbb{L}_e^{i,j} = \mathbb{L}_c^{i,j} \left[1 - \left(1 - \mathbb{L}_c^{i,j} \right)^{k_{i,j}} \right] \quad (19)$$

Simulation results in the previous section show that bufferless core networks can be unfair towards multi-hop TCP flows. Thus, to achieve fairness, multi-hop flows need more aggressive FEC than single-hop flows. We now capture this notion of fairness by formulating an optimisation problem as follows.

Inputs:

- Offered load $\lambda_{i,j}$ by every edge-to-edge flow (i, j).
- $r(i, j)$ the routing path of the flow (i, j).

Objective function:

$$\min_{k_{i,j}} \left(\max_{i,j} \mathbb{L}_e^{i,j} \right) \tag{20}$$

Subject to: $k_{i,j} \in \{1, 2, 3, \ldots\}$

Output: The set $\{k_{i,j}\}$, which denotes the optimum FEC block-size for every edge-to-edge flow.

The optimisation objective in Eq. (20) is a min-max objective expressed in terms of the edge-to-edge loss rate (conversely, it can be viewed as a max-min objective in terms of edge-to-edge goodput). It seems reasonable to consider the above objective since it ensures that the network bandwidth is assigned to the various (single- and multi-hop) flows in a fair manner, thus preventing the 1-hop flows from exploiting the available bandwidth and penalising the multi-hop flows (assuming that an ISP is equally concerned about multi-hop flows as single-hop flows).

It can be noted that the formulation has a non-linear objective function since the block-size $k_{i,j}$ that we are interested in is in the exponent of Eq. (19). In addition, we also have the constraint that each $k_{i,j}$ must be an integer. Although an optimal solution set exists for the formulation, the above two constraints render the problem intractable to large size networks. Indeed, the problem is NP-Hard since integer linear programming is itself NP-Hard. Thus in what follows, we propose a sub-optimal but practical heuristic that treats flows differently based on their hop-length.

5.4.2. HOP-LENGTH BASED SIMPLE HEURISTIC

In general, the best block-size to use for an edge-to-edge flow depends on the load on each of the links it goes through, which in turn depends on the FEC strength of the other flows traversing those links. To simplify these interdependencies and reduce the solution space, we consider a heuristic scheme in which flows of identical hop-length h use identical block-size k_h. For this assumption to be reasonable, two flows of similar hop-length should see similar link loads along their paths. One way this could hold is when all links in the network are roughly equally loaded, and further that the relative contribution to load by flows of different hop-lengths is similar across links. If loads vary significantly across links, one could generate a worst case bound in which all links are as heavily loaded as the maximum loaded link in the network. In this section, for purely illustrative purposes, we will assume that all links have the same average load λ and flows of different hop-lengths contribute equally to the load on each link. Specifically, for the NSFNet topology considered, flows are of 1-, 2- or 3-hops, and have offered load $\{\lambda_1, \lambda_2, \lambda_3\}$ given by

$$\lambda_1 = \lambda_2 = \lambda_3 = \frac{\lambda}{3} \tag{21}$$

Denoting $\overline{\lambda_c}$ to be the load on any core link c post FEC, then considering the 1-, 2- and 3-hop flows, $\overline{\lambda_c}$ is given by

$$\overline{\lambda_c} = \left(\frac{k_1+1}{k_1}\right)\lambda_1 + \left(\frac{k_2+1}{k_2}\right)\lambda_2 + \left(\frac{k_3+1}{k_3}\right)\lambda_3 \tag{22}$$

Employing the Poisson assumption, the probability of loss $\mathbb{L}_c$ at a core link c can be expressed as

$$\mathbb{L}_c = 1 - e^{\overline{\lambda_c}}\left(1+\overline{\lambda_c}\right) \tag{23}$$

Since we are assuming that all core links have the same average load, and hence the same loss rate, the core path loss probability $\mathbb{L}_{c,h}$ for a h-hop flow can be expressed as

$$\mathbb{L}_{c,h} = \mathbb{L}_c h \tag{24}$$

Now, from Eq. (15) the edge-to-edge packet loss probability $\mathbb{L}_{e,h}$ for a h-flow is

$$\mathbb{L}_{e,h} = \mathbb{L}_{c,h}\left[1-(1-\mathbb{L}_{c,h})^{k_h}\right] \tag{25}$$

We are now interested in finding the sub-optimal set $\{k_1,k_2,k_3\}$ that minimises the maximum edge-to-edge loss $\mathbb{L}_{e,h}$. Assuming that each k_h can take a value between 1 and 100 (since larger block-sizes are generally not beneficial, as we observed earlier), it is easy to use a simple brute-force approach to determine this set, and using MATLAB, we are able to obtain the result in just a couple of seconds since we only have a million combinations to choose from. For the same simulation setting described in Section 5.3.2. (the average NSFNet link load λ for a bufferless network being $\approx 11\%$), the resulting solution is k_1 = 19, k_2 = 4 and k_3 = 2. These results clearly suggest that we need to have different block-sizes for flows with different hop-lengths, and indeed multi-hop flows require a much more aggressive FEC scheme than single-hop flows (since $k_3 < k_1$).

5.4.3. SIMULATION RESULTS AND FAIRNESS ON THE NSFNET NETWORK

Using k_1 = 19, k_2 = 4 and k_3 = 2 (obtained from our heuristic in the previous subsection), we repeated the simulation with identical settings as before (having the same number of TCP flows on the NSFNet topology). Recall that our objective is to minimise the maximum edge-to-edge loss, or equivalently, maximise the minimum edge-to-edge goodput so as to achieve fairness across all flows. We employ the widely used Jain's fairness index [39] as an indicator of the heuristic's performance. It is extremely difficult to determine analytically what the optimum average goodput will be for the various 1-, 2- and 3-hop TCP flows from an overall network perspective, since analysing a single TCP flow is by itself very notorious, and we consider several thousand TCP flows with heterogeneous RTTs flowing through the network. Therefore as our benchmark, we again choose the goodput numbers obtained when all core links have delay-bandwidth buffers. If $\lambda_1^{'}$, $\lambda_2^{'}$ and $\lambda_3^{'}$ are the goodputs of 1-, 2- and 3-hop flows with large buffers, then the Jain's fairness index FI can be expressed as

$$FI = \frac{\left(\sum_{i=1}^{3}\frac{\lambda_i}{\lambda_i^{'}}\right)^2}{3\left[\sum_{i=1}^{3}\left(\frac{\lambda_i}{\lambda_i^{'}}\right)^2\right]} \tag{26}$$

Network setting	Average goodput (Mbps)			Fairness Index
	1-hop flows	2-hop flows	3-hop flows	
No FEC	1.571	0.667	0.391	0.78
k = 3 for all flows	2.219	0.807	0.397	0.70
k_1 = 19, k_2 = 4, k_3 = 2	1.090	0.760	0.596	0.96
delay - bandwidth buffers	1.509	1.440	1.359	1

Figure 29. Relative fairness indices

The fairness index is a real number between 0 and 1, with a higher value indicating better fairness. Fig. 29 shows the fairness index for four pertinent cases - no FEC, FEC with $k = 3$ for all flows, FEC with $k_1 = 19$, $k_2 = 4$ and $k_3 = 2$, and with delay-bandwidth core buffers.

Following are the salient observations we can draw from the figure. Firstly, the non-FEC case has a rather low fairness index of 0.78, and although setting the block-size to $k = 3$ improves the performance of 1- and 2-hop flows quite significantly, it fails to lift the goodput of 3-hop flows, and surprisingly performs poorly on the fairness scale when compared to the non-FEC case (0.70 compared to 0.78). Secondly, tuning the block-size according to our heuristic, corresponding to $k_1 = 19$, $k_2 = 4$ and $k_3 = 2$, results in a good fairness index of 0.96, confirming that the network bandwidth is indeed used by the various 1-, 2- and 3-hop flows efficiently. These results provide valuable insight on how a future bufferless core network can be envisaged using the edge-to-edge FEC scheme.

5.4.4. SENSITIVITY ANALYSIS OF THE BLOCK-SIZES

We now undertake a sensitivity analysis to ascertain how a slight perturbation to the values of k_1, k_2 and k_3 affects the fairness index. Our heuristic suggests using $k_1 = 19$, $k_2 = 4$ and $k_3 = 2$. We vary each k_h such that k_1 takes on values between 17 and 21, k_2 is in the range $\{3,4,5\}$, and k_3 is either 2 or 3. The simulation is repeated for each combination of the respective block-size. We note from the results that the fairness index varies between 0.924 (lower by 3.75%) and 0.962 (higher by 0.21%) when compared to 0.96 that was obtained by the heuristic, suggesting that the block-sizes derived by our heuristic algorithm is stable.

Before concluding this section, we wish to highlight that we performed many simulations by varying the distribution of the number of 1-, 2- and 3-hop TCP flows. The results we obtained follow closely the ones we have reported, omitted for brevity.

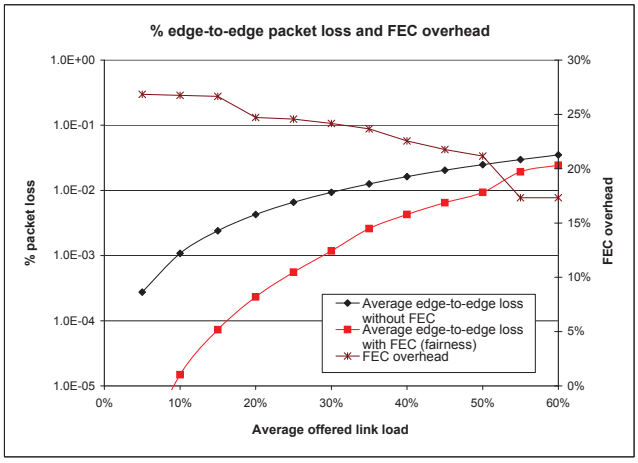


Figure 30. % edge-to-edge packet loss and FEC overhead for different average offered link loads obtained from analysis

5.4.5. PERFORMANCE OF THE FAIRNESS HEURISTIC AT DIFFERENT LOADS

The objective of this subsection is to provide some insights for when the proposed FEC fairness heuristic performs best. In Fig. 30, the Y axis on the left (note the log-scale) shows the average edge-to-edge packet loss (of 1-, 2- and 3-hops) as a function of the average offered link load for two scenarios, namely, without FEC and when we use the fairness heuristic. The Y axis on the right shows the overhead (in terms of load) that FEC introduces for a given average offered load. The non-FEC loss rates are obtained from Eq. (24), where the value of $\mathbb{L}_c$ for a given load is derived from Eq. (13). The loss rates with FEC are as a result of the heuristic.

There are essentially two main points that we want to make regarding the figure. First, if the ISP wishes to operate the network such that the edge-to-edge loss is below a certain acceptable threshold (say 10^{-3}), then we note that without FEC, the load can be pushed to at most 10%. On the other hand, employing the FEC heuristic allows the network to be run at 30% load, thus contributing significantly to the ISPs revenue stream. Second, operating the network at high load restrains FEC because there is not much room to introduce additional redundancy (as the total load cannot exceed the available capacity). Thus the benefit offered by the heuristic seems to diminish at higher loads ($> 60\%$). If the network is very lightly loaded ($< 10\%$), then loss rates are significantly low to begin with that there really is no incentive to use FEC. Since ISPs typically operate their networks at 20-30% load [35], the use of FEC in a future bufferless core network seems valuable.

These numbers must only be viewed in light of the proposed heuristic (that does not model TCP feedback and uses static block-sizes) since we could design more efficient TCP aware adaptive FEC schemes that adjust the block-sizes dynamically based on the loss rate that an edge-to-edge flow observes in its path. In general, FEC seems beneficial only under certain load regimes and not across the entire spectrum.

5.5. SECTION SUMMARY

As buffering of packets in the optical domain is a complex and expensive operation and we are soon approaching the limitations of electronics, we addressed the feasibility of enabling a wide-area bufferless (near-zero buffer) core optical network. We proposed a novel edge-to-edge based packet-level FEC architecture as a means of battling high core losses. We also considered a realistic core network (NSFNet), developed an optimisation framework, and proposed a simple heuristic to improve fairness between single- and multi-hop flows. The technique we have developed is a first step towards understanding the potential of FEC in envisioning a future bufferless core network.

6. CONCLUSION

In this chapter we first traced the evolution in thinking over the past few years on how much buffering is required at Internet routers. We then highlighted some of the implications of the move towards (potentially all-optical) KiloByte buffers, such as the reaction of TCP to reduced buffer availability in the network, and the unexpected interactions between TCP and real-time (UDP) traffic. Finally, we proposed mechanisms ranging from edge traffic pacing to packet-level forward error correction as means of overcoming the limitations posed by small buffers in the network.

We believe that through our ongoing work, we have only scratched the surface of what could well be a clean-and-green Internet architecture with near-zero buffers. “Green networking" has gained significant attention from ISPs and router manufacturers alike. There is scope for breaking new ground towards this goal. First, in terms of developing analytical and experimental models to understand TCP’s reaction to very small buffers. If there is further insight into the Poisson convergence, then its implications on the design of a future all-optical packet switched network can be huge. Second, extending the edge pacing and FEC frameworks to incorporate traffic grooming and load-balanced switching can aid ISPs in tuning and optimising their networks effectively. Finally, new service models appropriate to near-bufferless networks can be developed taking into account economics, power consumption and other user requirements.

References

[1] C. Villamizar and C. Song, “High Performance TCP in ANSNet,” *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 45–60, Oct 1994.

[2] P. Bernasconi et al., “Architecture of an Integrated Router Interconnected Spectrally (IRIS),” in *Proc. IEEE High Performance Switching and Routing*, Poland, 2006.

[3] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing Router Buffers,” in *Proc. ACM SIGCOMM*, USA, 2004.

[4] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, “Routers with Very Small Buffers,” in *Proc. IEEE INFOCOM*, Spain, 2006.

[5] N. Beheshti et al., "Buffer Sizing in All-Optical Packet Switches," in *Proc. IEEE/OSA OFC/NFOEC*, USA, 2006.

[6] D. Wischik and N. McKeown, "Part I: Buffer Sizes for Core Routers," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 75–78, Jul 2005.

[7] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing," in *Proc. IEEE INFOCOM*, Israel, 2000.

[8] R. S. Prasad, C. Dovrolis, and M. Thottan, "Router Buffer Sizing Revisited: The Role of the Output/Input Capacity Ratio," in *Proc. ACM CoNEXT*, USA, 2007.

[9] A. Vishwanath and V. Sivaraman and M. Thottan, "Perspectives on Router Buffer Sizing: Recent Results and Open Problems," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 2, pp. 34–39, Apr 2009.

[10] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," in *Proc. ACM SIGCOMM*, USA, 1995.

[11] "Cisco white paper: Approaching the zettabyte era http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481374.pdf, jun 2008."

[12] T. Karagiannis et al., "A Nonstationary Poisson View of Internet Traffic," in *Proc. IEEE INFOCOM*, Hong Kong, Mar 2004.

[13] D. R. Cox and V. Isham, "Point Processes," Chapman and Hall, 1980.

[14] J. Cao and K. Ramanan, "A Poisson Limit for Buffer Overflow Probabilities," in *Proc. IEEE INFOCOM*, USA, 2002.

[15] J. Cruise, "Poisson Convergence, in Large Deviations, for the Superposition of Independent Point Processes," *Submitted to Annals of Operations Research*, 2009.

[16] H. Jiang and C. Dovrolis, "Why is the Internet Traffic Bursty in Short Time Scales?" in *Proc. ACM SIGMETRICS*, Canada, Jun 2005.

[17] P. Borgnat et al., "Seven Years and One Day: Sketching the Evolution of Internet Traffic," in *Proc. IEEE INFOCOM*, Brazil, Apr 2009.

[18] A. Lakshmikantha, R. Srikant, and C. Beck, "Impact of File Arrivals and Departures on Buffer Sizing in Core Routers," in *Proc. IEEE INFOCOM*, USA, 2008.

[19] "ns-2 network simulator - http://www.isi.edu/nsnam/ns/."

[20] W. Feng, F. Chang, W. Feng, and J. Walpole, "A Traffic Characterization of Popular On-Line Games," *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 488–500, Jun 2005.

[21] "Packet traces from measurement and analysis on the WIDE Internet backbone. http://tracer.csl.sony.co.jp/mawi."

[22] V. Markovski, F. Xue, and L. Trajkovic, "Simulation and Analysis of Packet Loss in Video Transfers Using User Datagram Protocol," *The Journal of Supercomputing*, vol. 20, no. 2, pp. 175–196, 2001.

[23] L. Andrew et al., "Buffer Sizing for Nonhomogeneous TCP Sources," *IEEE Communications Letters*, vol. 9, no. 6, pp. 567–569, Jun 2005.

[24] G. Raina and D. Wischik, "Buffer Sizes for Large Multiplexers: TCP Queueing Theory and Instability Analysis," in *Proc. EuroNGI*, Italy, 2005.

[25] A. Vishwanath and V. Sivaraman, "Routers with Very Small Buffers: Anomalous Loss Performance for Mixed Real-Time and TCP Traffic," in *Proc. IEEE International Workshop on Quality of Service (IWQoS)*, Netherlands, 2008.

[26] A. Vishwanath and V. Sivaraman and G. N. Rouskas, "Considerations for Sizing Buffers in Optical Packet Switched Networks," in *Proc. IEEE INFOCOM*, Brazil, 2009.

[27] ——, "Anomalous Loss Performance for Mixed Real-Time and TCP Traffic in Routers with Very Small Buffers," *IEEE/ACM Transactions on Networking (pending minor revision)*, 2010.

[28] A. Vishwanath and V. Sivaraman, "Sharing Small Optical Buffers Between Real-Time and TCP Traffic," *Elsevier Optical Switching and Networking*, vol. 6, no. 4, pp. 289–296, Dec 2009.

[29] J. D. Salehi, Z.-L. Zhang, J. Kurose, and D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements Through Optimal Smoothing," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 397–410, Aug 1998.

[30] V. Sivaraman and D. Moreland, and D. Ostry, "A Novel Delay-Bounded Traffic Conditioner for Optical Edge Switches," in *Proc. IEEE High Performance Switching and Routing*, Hong Kong, May 2005.

[31] M. H. Overmars and J. van Leeuwan, "Maintenance of Configuration in the Plane," *Journal of Computer and System Sciences*, 1981.

[32] V. Sivaraman and H. Elgindy and D. Moreland and D. Ostry, "Packet Pacing in Short Buffer Optical Packet Switched Networks," in *Proc. IEEE INFOCOM*, Spain, 2006.

[33] ——, "Packet Pacing in Small Buffer Optical Packet Switched Networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1066–1079, Aug 2009.

[34] J. D. LeGrange et al., "Demonstration of an Integrated Buffer for an All-Optical Packet Router," *IEEE Photonic Technology Letters*, vol. 21, no. 2, pp. 781–783, Jun 2009.

[35] A. Odlyzko, "Data Networks are Mostly Empty and for Good Reason," *IT Pro*, vol. 1, no. 2, pp. 67–69, Mar/Apr 1999.

[36] N. Dukkipati and N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, Jan 2006.

[37] S. K. Korotky, "Network global expectation model: A Statistical Formalism for Quickly Quantifying Network Needs and Costs," *IEEE/OSA Journal of Lightwave Technology*, vol. 22, no. 3, pp. 703–722, Mar 2004.

[38] M. Mathis and J. Semke and J. Madhavi and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, Jul 1997.

[39] R. Jain and D. Chiu and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System," *DEC Technical Report*, 1984.