Delay-based Congestion Control: Sampling and Correlation Issues Revisited

G.D. McCullagh, D.J.Leith
Hamilton Institute, Ireland

***Abstract*— In this paper we revisit the commonly voiced concern that low correlation between measured delay and network congestion means that delay may be fundamentally flawed as a signal for congestion control. Our main contribution is to demonstrate that in fact what matters for congestion control is the *aggregate* behaviour of the flows sharing a link. Hence, perhaps somewhat surprisingly, while any given single flow may measure delay which is only weakly correlated with network congestion, this is not in itself an obstacle to congestion control.**

I. Introduction

In this paper we revisit the recently voiced concern that low correlation between measured delay and network congestion means that delay may be fundamentally flawed as a signal for congestion control. A related concern is that on heavily multiplexed links the measured delay may be only weakly correlated with the congestion window of a flow [10]. These concerns are particularly topical in view of a number of proposals to change the TCP congestion control algorithm to make use of delay information. Examples include not only FAST TCP [7] but also hybrid congestion control algorithms based on the use of both loss and delay, e.g. TCP Illinois [8] and Compound TCP [15]. The latter is now available in Windows Vista and is currently undergoing review at the IRTF and IETF standards bodies.

A number of recent independent measurement studies [1], [9], [11] have indeed found that there may be only low correlation between packet loss events and the delay measured by a flow. That is, when packet loss occurs, and thus some network queue is full, nevertheless all flows need not observe high delay. In fact, any given TCP flow may observe high delay at only a small proportion of packet loss events. This behaviour is confirmed by our own experimental measurements.

[9], [10] consider possible reasons for the low correlation observed and suggest that sampling issues are a fundamental factor. To see this, consider Figure 1 which presents an example queue occupancy time history at a bottleneck link carrying many flows. Packets from a single flow "sample" the queueing delay at the bottleneck link. However, when many flows share a link, the proportion of queued packets that are associated with a given flow can become small. The queueing delay is then only very sparsely sampled by that flow – for example, the "samples" for one flow are marked by solid squares in Figure 1. As a result, the flow cannot accurately estimate the state of the queue and may fail to detect even large changes in queue occupancy and, in particular, may fail to detect queue full events. Thus, in general, the correlation between the queueing delay measured by a given flow and congestion at a bottleneck link may be low.

This work was supported by Cisco Systems and Science Foundation Ireland grant IN3/03/I346.

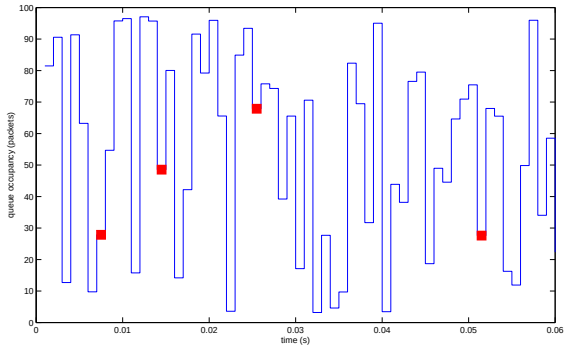


Fig. 1. Illustrating sampling issues when estimating queue state – example queue occupancy time history with packets of one selected flow marked by solid squares.

While the possibility of low correlation between network congestion and the delay measured by a flow thus seems well established, our aim in this paper is to investigate the implications for congestion control. A natural concern is that low correlation between measured delay and congestion means that delay measurements are *prima facie* an inadequate indicator of network congestion and thus their use for congestion control could be fundamentally flawed. Indeed, precisely such concerns are raised in [9], [10], [11]. This potentially has direct implications not only for recent delay-based proposals (FAST, Compound TCP etc) but also for our understanding of the fundamental constraints on congestion control within the current Internet architecture.

Our main contribution in this paper is to demonstrate that in fact what matters for congestion control is the *aggregate* behaviour of the flows sharing a link. Hence, perhaps somewhat surprisingly, while any given single flow may measure delay which is only weakly correlated with the congestion on a link, this is not in itself an obstacle to congestion control. In this paper we demonstrate this constructively via detailed experimental tests and also confirm analytically the general nature of our conclusions.

It is important to emphasise that this result does not preclude the existence of other factors that may limit the practical application of delay-based congestion control, it only states that low correlation is not itself an obstacle. For example,

obtaining good delay measurements in the presence of "noise" such as delayed acking and hardware offload is also potentially an important issue – in fact we touch on this issue here although it is not the main focus of the present paper. Despite the existence of such issues that require further study, we nevertheless argue that the work here is an important first step in exploring the nature of fundamental constraints on congestion control.

The paper is organised as follows. In Section III we briefly review a delay-based modification to the standard TCP AIMD algorithm that will be used later. In Section IV we consider the requirement for correlation between delay and loss events, analyse the congestion control properties of the delay-based AIMD algorithm and establish conditions under which it achieves congestion control, in the sense of bounding the queue occupancy. Section V presents experiments to validate this analysis, including in heavily multiplexed regimes and in the presence of "noise" such as delayed acking and TSO. We briefly summarise our conclusions in Section VII.

II. Related Work

Interest in the use of delay for congestion control is long-standing. Early work on delay-based algorithms includes CARD [6], Tri-S [16], and DUAL [17]). TCP Vegas ([2], [3]) and related algorithms such as FAST [7] are one of the most widely studied delay-based transport layer protocols. More recently, hybrid congestion control algorithms based on the use of both loss and delay have been proposed, e.g. TCP Illinois [8] and Compound TCP [15]. Compound TCP is being actively considered for use in a mainstream operating systems and is currently undergoing review at the IRTF and IETF.

In parallel with work on the development of delay-based algorithms, a number of concerns have been raised as to practicality of delay as a congestion signal. Potentially one of the most serious is the observation that there can be low correlation between measured and actual queueing delay and loss. This is discussed in detail in the experimental studies in [1], [9], [11]. These studies do not investigate any specific delay-based algorithm but rather make use of experimental measurements to evaluate correlation. [10] considers a number of possible reasons for low correlation to occur, including in particular sampling issues.

III. Delay-based AIMD

We begin by briefly reviewing the delay-based AIMD algorithm [4] which will be used later. Note that our purpose here is not to advocate use of the delay-based AIMD algorithm. Rather the delay-based AIMD algorithm is simply one approach to delay-based congestion control that happens to provide a useful vehicle for demonstrating some fundamental issues in a concrete manner. The algorithm simply extends the standard TCP AIMD algorithm to use delay as well as loss (or ECN) to control network congestion. Standard TCP employs an *Additive-Increase Multiplicative-Decrease* (AIMD) strategy during its congestion avoidance mode. AIMD congestion control can be implemented using signals other than packet loss as a congestion indicator. The basic idea here is that by backing off when queueing delay exceeds some threshold, we can avoid filling the queue (maintain low queueing delay) while staying within the well-established AIMD framework. Specifically, we consider the following delay-based AIMD algorithm:

$$cwnd \leftarrow \begin{cases} cwnd + \alpha/cwnd, & on\ each\ ACK \\ \beta \times cwnd, & if\ \tau \geq \tau_0 \\ \beta \times cwnd, & if\ packet\ loss \end{cases}$$

where τ is the observed queueing delay, $\tau_0 > 0$ is a delay threshold that triggers delay-based backoff. The queueing delay τ is estimated as $sRTT(t) - RTT_{min}$ where RTT_{min} is the minimum observed packet round-trip time and $sRTT(t)$ is an estimate of the current round-trip time. Loss induced backoffs are retained as part of the algorithm to accommodate situations where, for example, the buffers are sized such that maximum queueing delay is less than τ_0. Since it continues to employ an AIMD strategy, the delay-based algorithm inherits the usual fairness and convergence properties of AIMD.

The impact of this change on the AIMD operation is illustrated in Figure 3. It can be seen that although the buffer is sized at 400 packets, the flow $cwnd$ now backs off before the queue is full. In this example we can also see that following the first backoff where $cwnd$ is reduced by half, the queue empties for a significant period of time – this is to be expected as the delay-based algorithm backs off $cwnd$ before the queue is full. High utilisation be maintained by adjusting the backoff factor appropriately. In more detail, we adjust the backoff factor according to

$$\beta = RTT_{min}/RTT_{backoff}$$

where $RTT_{backoff}$ denotes the measured RTT at backoff. See [4],[14] for further details.

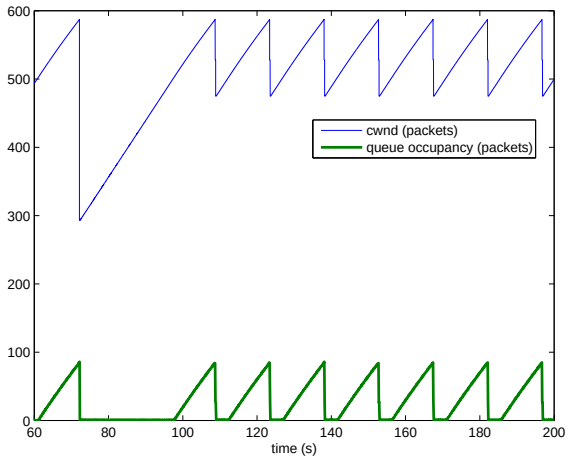


Fig. 2. Illustrating delay-based AIMD algorithm. (delay 120ms, link rate 50Mbps, 400 packet queue, τ_0 20ms, ns simulation).

IV. Is low correlation an obstacle to congestion control ?

On the face of it, the existence of situations where low correlation exists between the queueing delay measured by

a flow and the actual queueing delay appears to create an obstacle to congestion control. That is, how can we expect to succeed at congestion control if some flows are unable to reliably detect congestion events where the queue occupancy is high. We demonstrate that in fact what matters for congestion control is the *aggregate* behaviour of the flows sharing a link. Hence, while any given single flow may measure delay which is only weakly correlated with the actual queueing delay, this is not in itself an obstacle to achieving congestion control.

To explore this question we use the delay-based AIMD algorithm as an example and investigate its congestion control behaviour in more detail. We repeat again that our purpose here is not to advocate use of the delay-based AIMD algorithm. Rather the delay-based AIMD algorithm is simply one approach to delay-based congestion control that happens to provide a useful vehicle for demonstrating some fundamental issues in a concrete manner. Our analysis makes use of the following basic observation.

Key Observation. Sparse sampling of the queue occupancy means that packets from some flows may not detect an event where the queueing delay rises above a threshold τ_0. Nevertheless, if the queueing does rise above τ_0 then we *always* have that some packets do experience queueing delay greater than τ_0 – namely, the very packets that are responsible for filling the queue above threshold τ_0.

We immediately have that as long as the queueing delay remains above threshold τ_0, then each RTT the delay-based AIMD algorithm will lead to at least one flow backing off. Moreover, since every packet participating in the overshoot in queue occupancy above τ_0 measures the high queueing delay, the magnitude of the aggregate flow backoff is roughly proportional to the overshoot in queue occupancy (we make this statement more precise below). Intuitively, this creates pressure to drain the queue occupancy below τ_0, thereby achieving congestion control. This argument does not require that every flow be able to detect events when the queueing delay becomes high, it only requires that in aggregate the flows respond to each such congestion event. By the foregoing key observation, the latter is always the case. We develop this argument in more detail next.

A. Congestion Control Analysis

Consider n delay-based AIMD flows sharing a common bottleneck. Let w_i, t_i be the respective cwnd and round-trip propagation delay of flow i. Let $W = [w_1, \cdots, w_n]$, $T = [t_1, \cdots, t_n]$. Time k corresponds to the k'th congestion event, i.e. a network event where at least one flow backs off its cwnd. Figure 3 illustrates an example cwnd and queue time history. From the AIMD algorithm we have that

$$w_i(k+1) = \beta_i(k) w_i(k) + \alpha_i T_i(k)$$

where α_i is the AIMD increase parameter in packets/RTT, β_i is the AIMD backoff factor with $\beta_i(k) = 1$ corresponding to no back off of flow i at event k and $\beta \leq \beta_{max} < 1$ otherwise. $T_i(k)$ is the number of flow i RTTs between congestion events k and $k+1$.

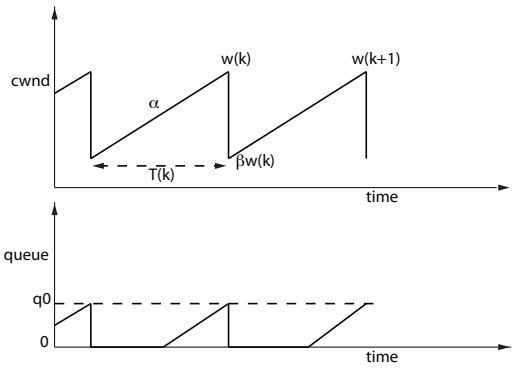


Fig. 3. Illustrating cwnd and queue time histories.

The queue occupancy $q(k)$ at the k'th congestion event is

$$q(k) = \sum_{i=1}^{n} [w_i(k) - b_i(k) t_i] \tag{1}$$

where $b_i(k)$ is the bandwidth being consumed by flow i and $\sum_{i=1}^{n} b_i(k) = B$ with B the link bandwidth in packet/s.

We let q_0 denote the queue occupancy corresponding to the backoff delay threshold τ_0. Note that this implicitly assumes that a one to one correspondence exists between queue occupancy and queueing delay. This assumption is satisfied, for example, for any link with constant service rate.

Also assume, for the moment, that a delay-based flow backs off its cwnd whenever one or more packets measures queueing delay above threshold τ_0. We return to this assumption later in Section IV-D. We then have the following identity:

$$\begin{aligned} \sum_{i=1}^{n} \beta_i(k) w_i(k) &\leq \beta_{max}(q(k) - q_0) + \sum_{i=1}^{n} w_i(k) - (q(k) - q_0) \\ &\leq \sum_{i=1}^{n} w_i(k) + (\beta_{max} - 1)(q(k) - q_0) \end{aligned} \tag{2}$$

Hence, combining (1) and (2),

$$\begin{aligned} q(k+1) &= \sum_{i=1}^{n} [w_i(k+1) - b_i(k+1) t_i] \\ &= \sum_{i=1}^{n} [\beta_i(k) w_i(k) + \alpha_i T_i(k) - b_i(k+1) t_i] \\ &\leq \beta_{max} q(k) + (1 - \beta_{max}) q_0 + \sum_{i=1}^{n} \alpha_i T_i(k) \\ &\quad + \sum_{i=1}^{n} [w_i(k) - b_i(k+1) t_i - q(k)] \end{aligned} \tag{3}$$

When flows are synchronised, in steady state[1] $b_i(k+1) = b_i(k)$ and the terms in the square brackets sum to zero. Hence,

[1]The existence of a unique, stable steady-state solution for synchronised AIMD-based TCP networks is shown in, for example, [13].

the queue length is constrained by

$$q(k+1) \quad \leq \quad \beta_{max} q(k) + (1-\beta_{max}) q_0 + \sum_{i=1}^{n} \alpha_i T_i(k)$$

Since $\beta_{max} < 1$ this recursion is convergent and so as $k \to \infty$

$$q(k) \quad \leq \quad q_0 + \sum_{i=1}^{n} \alpha_i T_i(k)/(1-\beta_{max}) \tag{4}$$

Since the time $T_i(k)$ between congestion events is necessarily bounded (trivially, since the network capacity is bounded the flow cwnds cannot increase indefinitely), we have that $q(k)$ is upper bounded.

The bound in (4) is potentially very conservative. Nevertheless, it establishes that a network of synchronised delay-based AIMD flows will always converge to operation with bounded queues and so achieve congestion control. This result holds even when the delay measured by any given single flow may be weakly correlated with queue excursions above the threshold τ_0. Thus we have established that low correlation is not in itself an obstacle to achieving congestion control. Of course we need to validate this theoretical analysis via experimental measurements and this is the subject of Section V. Before that, however, we first consider some important extensions of our analysis.

B. Two important cases

Tighter bounds on the queue occupancy can be readily obtained in two common cases.

Case 1: Queue occupancy falls below q_0 (i.e. below the delay threshold τ_0) on flow cwnd backoff following congestion event k. In the worst case (i.e. corresponding to peak queueing delay), congestion event $k+1$ occurs when all flows increase their cwnd's at the point where the queue is just below q_0. That is,

$$q(k+1) \leq q_0 + \sum_{i=1}^{n} \max(\alpha_i, 1) \tag{5}$$

The bound (5) has a natural interpretation. Due to network delays, no flow can detect an event where the queue exceeds τ_0 until at least one RTT after the event. During this RTT it can therefore happen that every flow increases its cwnd and injects additional packets into the network. As a result, the size of the resulting queue overshoot can, in the worst case, be proportional to the number n of flows.

Case 2: Queue occupancy remains above q_0 following congestion event k. In this case $T(k) \leq max_{i \in \{1,2,..,n\}} t_i$ and so

$$q(k) \quad \leq \quad q_0 + \sum_{i=1}^{n} \max(\alpha_i, 1)/(1-\beta_{max}) \tag{6}$$

C. Unsynchronised Flows

The foregoing analysis is for networks where flow backoffs are synchronised. The analysis can be readily generalised to situations where the flows are not synchronised. In more detail, taking expectations in (3) yields

$$E[q(k+1)] \leq \beta_{max} E[q(k)] + (1-\beta_{max}) q_0 + \sum_{i=1}^{n} \alpha_i E[T_i(k)] + \sum_{i=1}^{n} [E[w_i(k)] - E[b_i(k+1)] t_i - E[q(k)]]$$

In steady state[2] $E[b_i(k+1)] = E[b_i(k)]$, the terms in the square brackets sum to zero and length is constrained by

$$E[q(k+1)] \quad \leq \quad q_0 + \sum_{i=1}^{n} \alpha_i E[T_i(k)]$$

Thus our conclusion that low correlation is not an obstacle to congestion control remains unchanged in unsynchronised networks.

D. Filtered delay measurements

The foregoing analysis assumes that the delay-based AIMD algorithm backs off its cwnd whenever one or more packets measure queueing delay above threshold τ_0. However, in practice we expect to use a smoothed estimate of queueing delay in order to avoid backing off in response to spurious delays. In other words we expect to backoff cwnd only after a sufficient number of packets have experienced high queueing delay.

The choice of an appropriate smoothing filter is a design question that is outwith the scope of the present paper. Instead, our interest here is in understanding the impact such smoothing may have on the congestion control properties of a delay-based algorithm. One direct approach is to seek a class of filters under which the previous analysis remains applicable. It can be seen immediately that as long as we maintain the identity (2), then the general bound (4) remains unchanged. For example, this holds when we modify the based delay-based AIMD algorithm to be

$$cwnd \leftarrow \begin{cases} cwnd + \alpha/cwnd - \gamma, & on\ each\ ACK \\ \min[\beta(cwnd + S_\gamma), cwnd], & if\ \tau \geq \tau_0 \\ \beta \times cwnd, & if\ packet\ loss \end{cases}$$

where $\gamma > 0$ if the currently acknowledged packet has experienced queueing delay above τ_0 (this may be estimated via the packet time stamp), otherwise $\gamma = 0$. Thus the algorithm makes a small reduction in cwnd when individual packets experience high delay. This satisfies (2) by ensuring that cwnd is always backed off at least in proportion to the number of packets with high queueing delay. S_γ is a running total of γ values and is reset to zero on a full backoff (i.e. when $\tau \geq \tau_0$). It is therefore the overall amount, since the last full backoff, subtracted from cwnd due to individual delayed packets. We use S_γ to adjust the decrease in cwnd at a full backoff to avoid double counting of delayed packets. The delay signal τ triggering full backoff may be any smoothed estimate of queueing delay, thereby decoupling the baseline

[2]Under mild assumptions, the existence of a unique stationary distribution for unsynchronised TCP networks is shown in, for example, [12].

	Description
CPU	Intel Xeon CPU 2.80GHz
Memory	512 Mbytes
Motherboard	Dell PowerEdge 860
Kernel	Linux 2.6.18
txqueuelen	1,000
max_backlog	300
NIC	Intel 82540EM
NIC Driver	e1000 5.2.39-k2
TX & RX Descriptors	4096

TABLE I
HARDWARE AND SOFTWARE CONFIGURATION.

congestion control behaviour of the algorithm from the choice of smoothing filter.

The per packet backoff factor γ may differ from the standard backoff factor β. One natural choice is $\gamma = 1 - \beta$ which ensures back off to $\beta \times cwnd$ when a full windows worth of packets are delayed. Since we might expect to choose the smoothing filter such that τ exceeds τ_0 when a full cwnd of packets exceeds τ_0 (indeed, perhaps when less than a full cwnd of packets exceed τ_0), then the per packet backoff simply acts as a conservative "safety net" .

Again, we have that low correlation is not an obstacle to congestion control even when a filtered delayed signal is used.

V. EXPERIMENTAL MEASUREMENTS

To help build confidence in the validity of the foregoing analysis for real network traffic, in this section we explore delay-based congestion control behaviour using experimental measurements taken on a hardware testbed. Use of experimental tests seems particularly important in the context of delay-based control as issues such as scheduling granularity, hardware offload, packet bursts *etc* are difficult to model accurately yet may have a direct impact on delay measurements and performance.

This section is organised as follows. We begin by describing the experimental setup used and then demonstrate that low correlation between delay and loss is prevalent on heavily multiplexed links. Initially delayed acking is disabled, as is TCP segmentation offload (TSO) in order to focus on correlation issues. We explore the congestion control behaviour of the delay-based AIMD algorithm and compare experimental measurements with the analysis in Section IV. In Section V-D we discuss in more detail some observations on asymptotic behaviour as the number of flows becomes very large. In Section V-F we enable delayed acking and TSO and consider the impact on delay measurement and congestion control.

A. Experimental setup

We have implemented the delay-based AIMD algorithm in Linux 2.6.23. Experiments were carried out using a testbed consisting of commodity PCs connected to gigabit switches to form the branches of a dumbbell topology. All sender and receiver machines used in the tests have identical hardware and software configurations as shown in Table I and are connected to the switches at 1Gb/sec. The router, running FreeBSD v4 with the dummynet module, can be configured with various bottleneck queue-sizes, capacities and round trip propagation delays to emulate a range of network conditions. TCP Flows are injected into the testbed using `iperf`. TCP stacks are instrumented using a modified version of the Linux `tcpprobe` module. Unless otherwise stated, the queueing delay threshold used is τ_0=50ms. On receipt of an ACK packet an RTT measurement RTT is obtained by comparing the time that the ACK packet is received with the time that the corresponding data packet was transmitted. In Section V-B – Section V-E delayed acking and TSO is disabled. These are then enabled in Section V-F in order to explore their impact on congestion control behaviour. The minimum observed RTT measurement RTT_{min} is used as an estimate of propagation delay and queueing delay is then estimated as $RTT - RTT_{min}$.

B. Illustrating low correlation

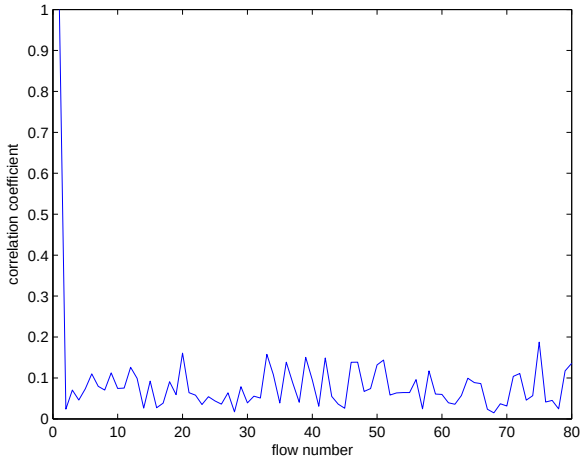
We begin by considering a heavily multiplexed link where we expect low correlation between delay and loss to be prevalent. Figure 4 demonstrates the congestion control action of the delay-based AIMD algorithm on a 10Mbps link shared by 80 TCP flows. The flows have a range of round-trip times from 20-200ms. Figure 4(a) plots the correlation between the queueing delay measured by flow 1 and the queueing delay measured by flows 2-80[3]. These results are representative, with similar correlation values obtained for flows other than flow 1. It can be seen that the correlation between measured queueing delay is close to zero for all flows. Figure 4(b) shows typical time histories of the measured queueing delay (i.e. measured RTT minus the known propagation delay), and the low level of correlation between measurements is evident.

Despite the low correlation between measured queueing delay, the delay-based algorithm successfully regulates the flow cwnds to prevent queue overflow and congestion. This is illustrated in Figure 5(a) which plots the sum of the flow cwnds while Figure 5(b) plots typical cwnd time histories for individual flows. During this test run no packet losses occurred. Note that the small flow cwnds in Figure 5(b) are feature of the link being highly multiplexed and reflect the fact that here we are intentionally seeking to explore situations where flow delay measurements may be weakly correlated.

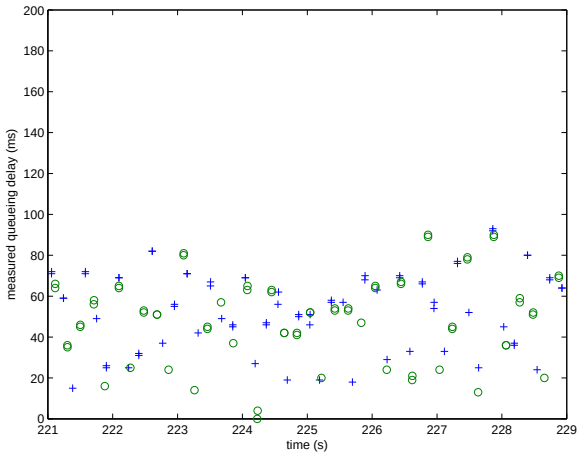
C. Peak queueing delay vs number of flows

To explore congestion control performance in more detail we consider the measured queueing delay as a function of the number of flows sharing a link. Figure 6 plots the mean and peak queueing delay as the number of competing flows is varied on a link. Also marked on Figure 6 is the analytic bound (5). This worst case bound captures the fact that no flow can detect a queue overshoot above τ_0 until one RTT after it occurs. Thus it can happen that all flows increase their cwnd and insert extra packets with one RTT, creating an overshoot above τ_0 that is proportional to the number of flows. It can

[3]Queueing delay is the measured from the packet time-stamps less $baseRTT$, where $baseRTT$ is the minimum observed delay. We confirmed that flow $baseRTT$ estimates of propagation delay were accurate. Time histories are aligned based on the peak correlation.

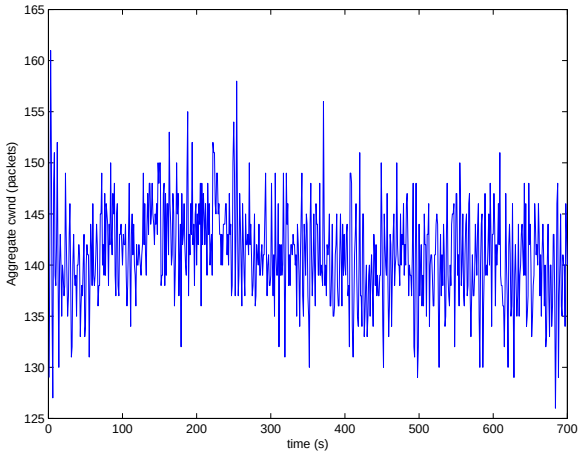


(a) Correlation between queueing delay measured by flow 1 and by flows 2-80.

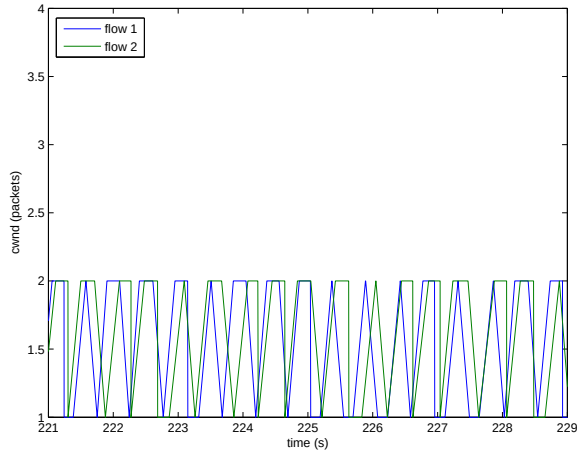


(b) Time histories of measured queueing delay for flow 1 and flow 2.

Fig. 4. 80 concurrent, long-lived flows. 10Mbps link, flow RTTs uniformly randomly chosen from 20-200ms.



(a) Sum of flow cwnds.



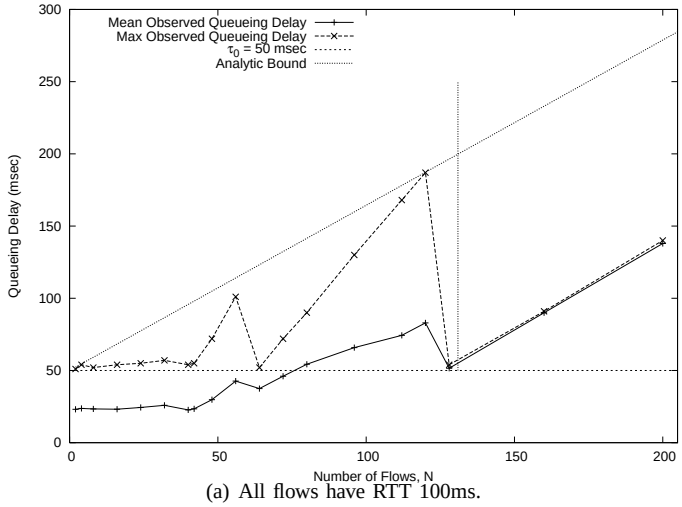
(b) cwnd time histories for flow 1 and flow 2.

Fig. 5. 80 concurrent, long-lived flows. 10Mbps link, flow RTTs uniformly randomly chosen from 20-200ms.

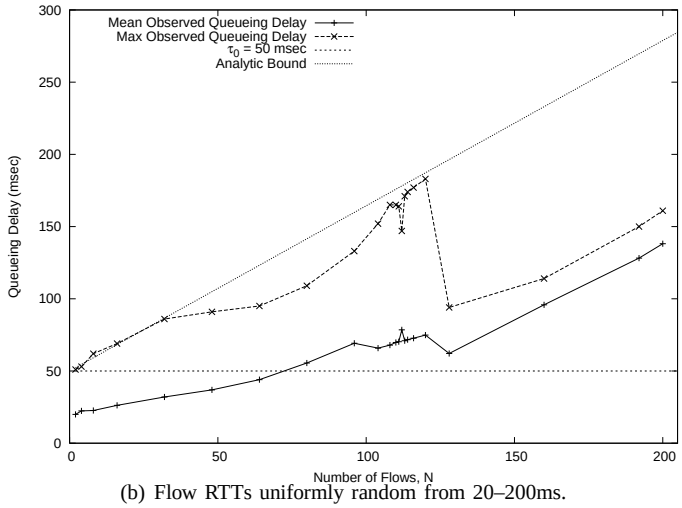
be seen that as we increase the number of flows the peak queueing delay the analytic bound is generally quite tight. This not only provides a degree of validation of the analysis in Section IV but also provides a concrete demonstration that a delay-based algorithm can indeed achieve effective congestion control under low correlation conditions.

Investigating these experimental results in more detail, it can be seen from Figure 6 (and Figure 8) that the peak queueing delay does not increase monotonically with the number of flows but rather may decrease as the number of flows increases. This arises due to quantisation of the number of packets in flight. To see this consider n flows sharing a link with bandwidth-delay product P packets and with queue occupancy q_0 packets corresponding to the queueing delay threshold τ_0. Assume, for simplicity, that the flows are perfectly synchronised and have the same cwnd. Let $\underline{cwnd}(n) = \max\{j : j \times n - P < q_0, j \in \{0, 1, 2, ...\}\}$. That is, $\underline{cwnd}(n)$ is the largest cwnd such that the queueing delay still remains below the threshold τ_0. Overshoot in queue occupancy above q_0 will then occur on the next round-trip time when flows increase their cwnd by one packet, with the magnitude of the overshoot being $(\underline{cwnd}(n) + 1) \times n - P - q_0$. As we increase the number of flows to $n+k$, then the flow cwnd just before backoff initially remains unchanged i.e. $\underline{cwnd}(n+k) = \underline{cwnd}(n)$, provided $\underline{cwnd}(n) \times (n+k) - P < q_0$. Eventually, however, as $n+k$ increases further $\underline{cwnd}(n+k)$ must reduce to be smaller than $\underline{cwnd}(n)$. At this point the delay overshoot will also decrease, leading to non-monotonic behaviour of the overshoot. It can seen from Figures 6(a) that, as might be expected, this effect is most pronounced when all flows have the same RTT and so are almost synchronised. When flows have different RTTs, as in Figure 6(b), and are unsynchronised the overshoot tends to show a simple rising trend.

Also plotted in Figure 6 is the mean queueing delay vs number of flows. It can be seen that the mean delay also displays a rising trend. This has implications for performance and operation at the "knee of the curve". The potential exists to modify the delay-based algorithm to mitigate this effect e.g.

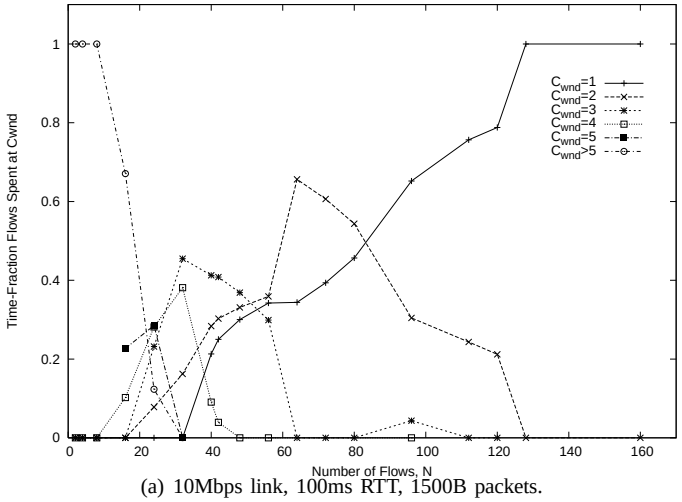


(a) All flows have RTT 100ms.

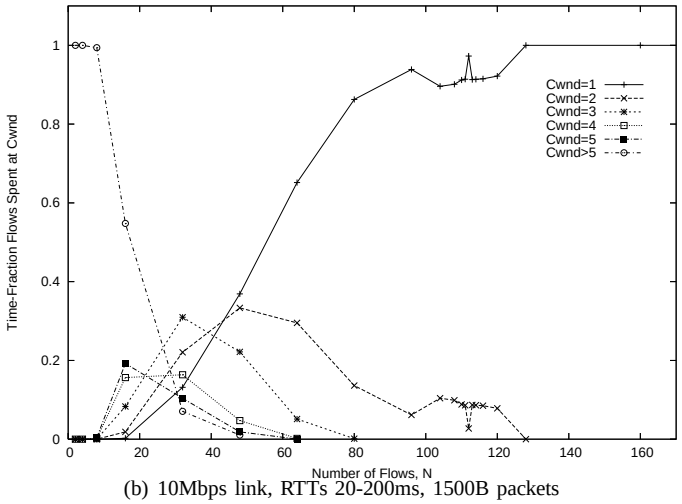


(b) Flow RTTs uniformly random from 20–200ms.

Fig. 6. Peak and mean queueing delay vs number of flows. 10Mbps link. Bound (5) is labelled "Analytic Bound". The vertical line in (a) marks the point at which the number of flows equals the path bandwidth-delay product.



(a) 10Mbps link, 100ms RTT, 1500B packets.



(b) 10Mbps link, RTTs 20-200ms, 1500B packets

Fig. 7. Fraction of time that flows take values of $cwnd$ vs the number of flows. It can be seen that flow $cwnd$'s tend to decrease as the number of flows increases, until all flows have $cwnd$ of one packet.

by adjusting the threshold τ_0 based on observed overshoot in delay. However, we do not explore this possibility here.

D. Asymptotic behaviour

While our experimental results demonstrate that low correlation need not be an obstacle to congestion control, they also highlight that other issues can arise on links with very large numbers of flows. In particular, as the number of flows is increased the flow cwnd's will eventually decrease until they are only one packet in size. This is inevitable since newly added flows must have a cwnd of at least one packet and therefore existing flows must, if possible, reduce their cwnd to make space for new flows (or rather backoff their cwnds to maintain the queueing delay below τ_0). Eventually, however, we will reach the situation where all flows have reduced their cwnd to one packet.

This behaviour is illustrated in Figure 7, which plots the flow cwnds as the number of flows is increased. It can be seen that as the number of flows is increased the flow cwnds tend to fall, until eventually all flows have cwnd of one packet. The point where this occurs can be calculated as follows. In Figure 7(a) flows with a base RTT of 100ms share a 10Mbps link. The number of packets required to fill the pipe and create a queueing delay of τ_0 is $B(RTT+\tau_0)$, where B is the link rate in packets per second . With τ_0=50ms and 1500 byte packets, $B(RTT + \tau_0)$=130 packets i.e. our limit is 130 flows with cwnd of 1 packet. This is confirmed by inspection of Figure 7(a). In the mixed base RTT case shown in Figure 6(b) it is harder to analytically calculate the number of flows where the limiting regime occurs. Nevertheless, the qualitative behaviour is similar as can be seen from Figure 7(b).

In this asymptotic regime, each additional new flow leads to an increase in the level of queue occupancy. This occurs despite the fact that the queueing delay may then remain persistently above τ_0, since flows all have cwnd of one packet and so cannot backoff their cwnd further to reduce the queueing delay. Note also that when the delay is persistently above τ_0, delay-based AIMD flows will not increase their cwnd. Hence, we have that the queueing delay simply increases linearly with the number of flows. This behaviour is evident in Figure 6(b) when the number of flows is greater than 130 – it can be seen

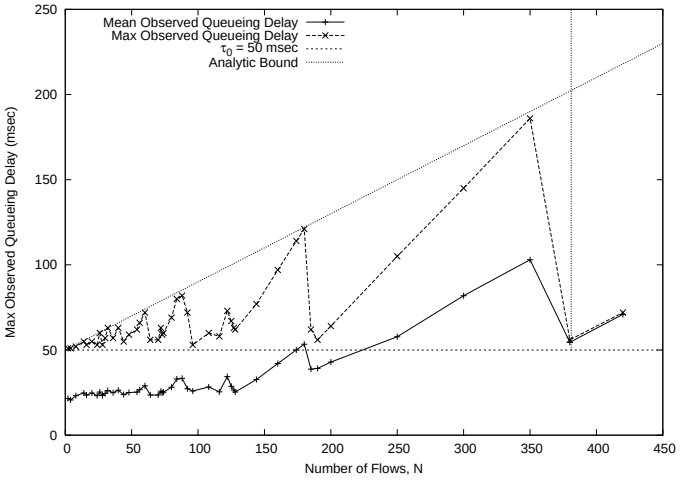


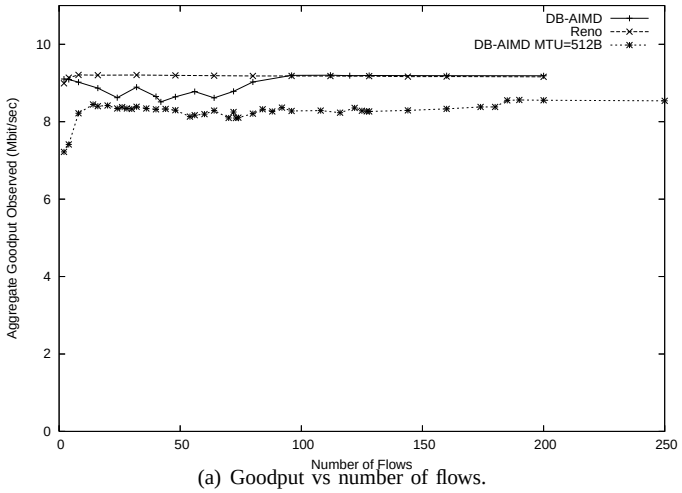
Fig. 8. Peak and mean queueing delay vs number of flows. 10Mbps link, 100ms RTT, 512B packets.

that the delay rises linearly, parallel to the analytic bound (5).

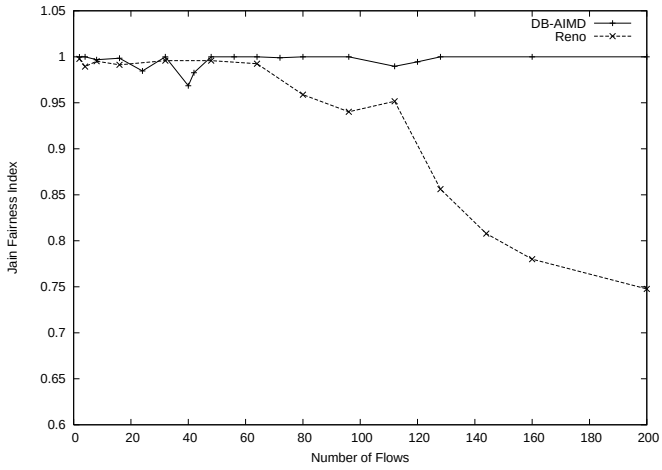
We discuss SACK Reno in more detail in the next section, but note here that once the cwnd of a flow falls below three packets, fast retransmit no longer operates (since it requires two duplicate acks after loss of one packet) and congestion control falls back retransmit timeouts (RTO). SACK Reno behaviour is the asymptotic regime is thus complex and, for example, prone to prolonged unfairness between competing flows due to sensitivity to loss of retransmitted packets when in RTO.

The fact that flow cwnds are constrained to have a minimum value of one packet appears to place a limit on the use of delay-based congestion control. Namely, as the number of flows is increased to the point where a flow cwnd of less than one packet is needed to maintain low queueing delay, then low delay operation becomes impossible and packet loss eventually occurs as the number of flows becomes sufficiently large.

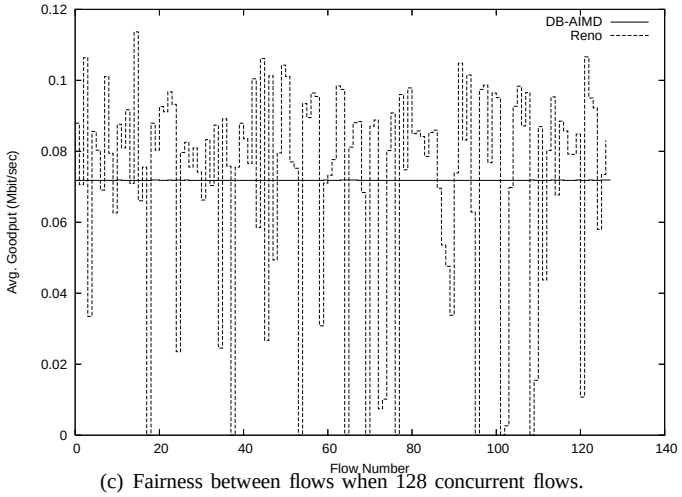
The limit is, however, not a fundamental one. For example, while cwnd is constrained to be at least one packet, we might reduce the packet size to prevent queue buildup. For example, we re-ran our experiments using a packet size of 512 bytes rather than 1500 bytes. Figure 8 shows the corresponding results – we can now fit more than 390 flows on the link before the flow cwnds are all reduced to only one packet. Reducing packet size may not be an attractive solution, however, as it increases the transmission overhead (packet headers and link framing overhead remain unchanged as the packet payload is decreased in size). One alternative is to insert delays between packet transmissions so as to pace packets at a slower rate, which would soften the impact of the one packet lower bound on cwnd. However, situations where this limit is reached are essentially corner cases where flows are all getting very low throughput and the user experience is likely to be poor regardless of changes to TCP. For example, on a 10Mbps link with 130 flows, each flow is has a throughput share of less than 75Kbps, i.e. similar to a dialup modem. With 512B packets, for $>$390 flows the per share is less than 25Kbps.



(a) Goodput vs number of flows.



(b) Fairness vs number of flows.

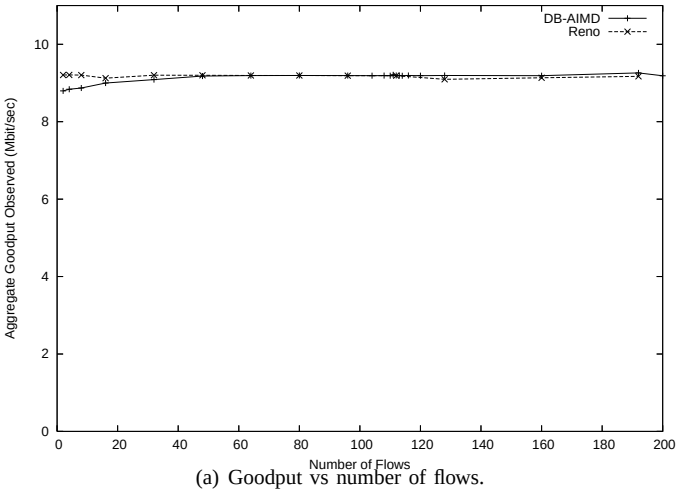


(c) Fairness between flows when 128 concurrent flows.

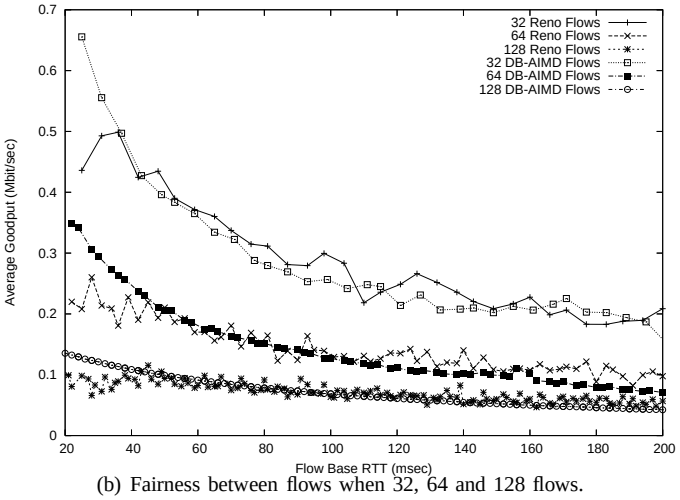
Fig. 9. Goodput and fairness for Reno and delay-based AIMD algorithms. 10Mbps link, 100ms RTT.

E. Comparison with SACK Reno

To illustrate that effective congestion control is indeed being achieved by the delay-based algorithm even when correlation is weak, it is informative to compare behaviour with that of the standard SACK Reno loss-based TCP. Figures 9 and 10 compare the performance of the delay-based algorithm with that of the standard Linux SACK Reno algorithm. Figure 9 show measurements when flows have the same base RTT of 100ms and Figure 10 shows measurements when the flow base RTTs are distributed between 20-200ms. It can be seen that for a given packet size link utilisation is at worst slightly reduced with delayed-based AIMD i.e. the low delay achieved by the delay-based AIMD algorithm does not come at the cost of a significant lowering of throughput. As discussed previously, the negative impact on throughput of using a smaller packet size is evident in Figure 9(a). While the aggregate link utilisation is similar, it can be seen from Figure 9(b) that the delay-based algorithm achieves better inter-flow fairness than Reno on heavily-multiplexed paths. It can also be seen from Figure 10(b) that the difference in fairness behaviour is less pronounced when there is wider mix of flow RTTs, and so of flow cwnds. At low cwnds, fast recovery is ineffective and congestion control in Reno reverts to RTO operation. Unfairness then arises because flows in RTO become very sensitive to packet loss – following an RTO, if the first retransmitted packet is lost then the RTO timer is doubled and can easily increase to several seconds duration. In contrast, the delay-based algorithm avoids packet loss even in quite extreme regimes.



(a) Goodput vs number of flows.



(b) Fairness between flows when 32, 64 and 128 flows.

Fig. 10. Goodput and fairness for Reno and delay-based AIMD algorithms. 10Mbps link, 20-200ms RTT.

F. Delayed ACKing and TSO

The foregoing experimental results disable delayed acking and TSO in order to focus clearly on the fundamental performance of the delay-based algorithm. In this section we consider the impact of delayed acking and TSO in more detail. Although our primary focus in this paper is on the impact of low correlation in congestion control, we found that our experimental tests also helped to throw some light on the issue of obtaining clean delay measurements. In particular, we found that some significant sources of "noise" in delay measurements can be removed by simple sender side changes to delay measurement within the network stack. These changes have since been incorporated into the Linux kernel.

Delayed acking can be a source of error in delay measurements. With delayed acking, a receiver does not immediately acknowledge receipt of a data packet but instead waits until either a second packet arrives or a timer expires (the timer may be a fixed delay e.g. 100 ms or might be adaptive e.g. in Linux the timer is adapted to be roughly related to the inter-packet spacing) before transmitting a TCP ACK packet back to the data sender. In practice delayed acking is almost universally used and creates additional delay within the end host receiver. This additional delay is unrelated to queueing delay and network conditions and so is a form of "noise" on the measured packet RTT. An obvious concern is that this "noise" may negatively impact delay-based congestion control.

In a similar vein, in this section we also consider the impact of hardware TCP segmentation offload (TSO). TSO is widely used in modern network cards, particularly at higher line rates. With TSO large sized segments are handed off to the network interface card for conversion into multiple packets before transmission. This can create additional delay while the end host sender waits for a sufficient backlog of enqueued data to form a large segment – a timer is typically used to bound the delay i.e. a segment is handed off to the hardware once sufficient packets have been accumulated for transmission or, failing that, when the timer expires. TSO can also lead to increased burstiness of packet transmissions (a segments worth of packets are sent back to back) and thus perhaps to changes in queueing delay behaviour.

Figure 11 plots RTT measurements illustrating the impact of delayed acking. The data marked "timestamp" corresponds to the packet RTT measurements obtained as per RFC 1323 [5] – these are the standard RTT values available within the network stack. While originally introduced for retransmit timeout (RTO) calculations, these measurements are also used by many delay-based congestion control algorithms. Note the large spikes in the measured delay due to delayed acking. These arise because RFC 1323 mandates that RTT is measured

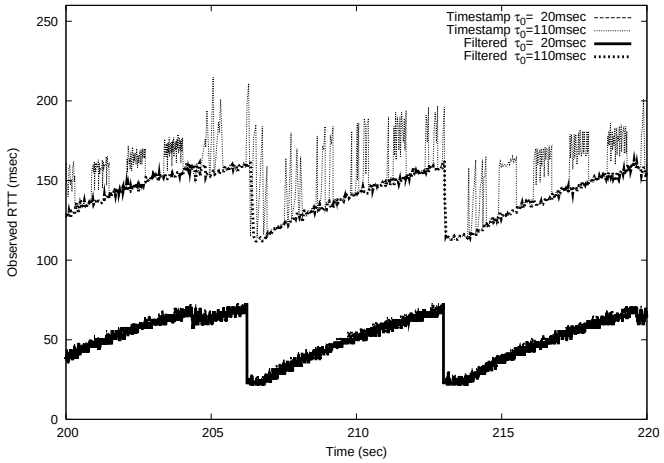


Fig. 11. Example time histories of measured RTT with delayed acking. "Timestamp" denotes the RTT measured as per RFC 1323. "Filtered" denotes RTT measured as the time between arrival of an ACK and transmission of the most recent packet acked. 10Mbps link, two flows, RTT of flow 1 is 20ms and of flow 2 is 110ms.

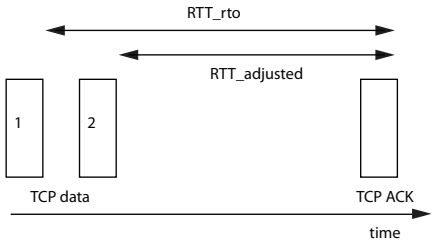


Fig. 12. Illustrating RTT calculations with delayed acking via timeline at sender. Data packet 2 generates a delayed ack at the receiver which arrives at the sender after a round-trip time. RTT_rto denotes the RTT measurement defined in RFC 1323 which includes both the network round-trip time and the delay due to delayed acking at the receiver, RTT_adjusted denotes the RTT measurement without delayed acking.

as the time between transmission of the *oldest* packet acked and arrival of the ACK packet. This is illustrated for example in Figure12.

It can be seen that RTT measured as per RFC 1323 includes the delay within a receiver before an ack is sent. While this makes sense for RTO calculations (inflating the RTO threshold and avoiding spurious timeouts being induced by delayed acking), it is less useful for delay-based congestion control since the delay within the receiver is unrelated to network congestion. Although we might use filtering to attenuate the delay spikes induced by delayed acking, effective filtering requires some form of averaging over multiple delay measurements. We note that the impact of delayed acking is particularly pronounced at low cwnds (e.g. in Figure 11 the impact is greatest on flow with longer RTT which has cwnd of around 10 packets rather than the short RTT flow which has cwnd around 40 packets) where delay measurements are relatively widely spaced due to the small number of packets in flight and filtering can be problematic. For example, if a flow has cwnd of one packet, delay measurements are taken an RTT apart and owing to the long delay between measurements it can be difficult to decide whether an increase in delay should be attributed to delayed acking or to network congestion.

Fortunately, a clean RTT signal without delayed acking delay can be readily obtained by a small change to the delay measurement calculation used for congestion control (the delay measurement used for RTO calculations is, of course, left unchanged). Specifically, by using an RTT measurement calculated as the time between transmission of the *newest* packet acked and arrival of the ACK packet – see Figure12. In addition, when delayed acking is detected we ignore the RTT value from ACK packets that ack a single packet (as opposed to acking multiple packets). These changes are straightforward to implement but their impact is substantial. This is illustrated, for example, in Figure 11 by the data marked "filtered". It can be seen that the delay spikes associated with delayed acking are completely removed. That is, by a small change to the delay measurement approach we are able to remove a significant source of "noise".

Figure 13 illustrates the impact of TSO on delay measurements. The data marked "2.6.18" shows measurements taken using the Linux 2.6.18 kernel and it can be seen that TSO can induce large spikes in the measured RTT. These spikes are associated with the kernel using a simple timer with the relatively large value of 40ms. A segment is handed off to the hardware once sufficient packets have been accumulated for transmission or, failing that, when the timer expires. The data marked "2.6.24" show measurements for the same experimental setup but now using a more recent 2.6.24 kernel that includes modifications (motivated in part by the work in this paper) that implement more sophisticated TSO handling. It can be seen that the spikes are almost completely removed. Also shown in Figure 13 is the queue occupancy time history from which it can be seen that the RTT measurement now has little noise.

With the changes to RTT measurement and TSO handling discussed above, Figure 14(a) plots the measured queueing delay vs number of flows with delayed acking and TSO enabled. In these tests we also make a change to the delay-based AIMD algorithm to prevent delay backoff reducing the cwnd below two packets (loss backoff and RTO behaviour is unchanged). This is because with delayed acking enabled we need at least two packets in flight in order to obtain a clean delay measurement using the approach discussed above. It can be seen from Figure 14(a) that the performance is similar to that without delayed acking and TSO – compare with Figure 6(b). In particular, the queueing delay remains bounded even for large numbers of flows where the correlation between measured and actual queueing delay is low. The point at which the link enters the asymptotic regime is changed, however, from around 130 flows to around 90 flows owing to the lower bound on cwnd of two packets in Figure 14(a) while in Figure 6(b) the lower bound is one packet.

Also shown in Figure 14(b) is the corresponding measured link goodput. It can be seen that the goodput is almost identical to that without delayed acking and TSO. Although not shown here, throughput fairness between flows is also very similar to that without delayed acking and TSO.

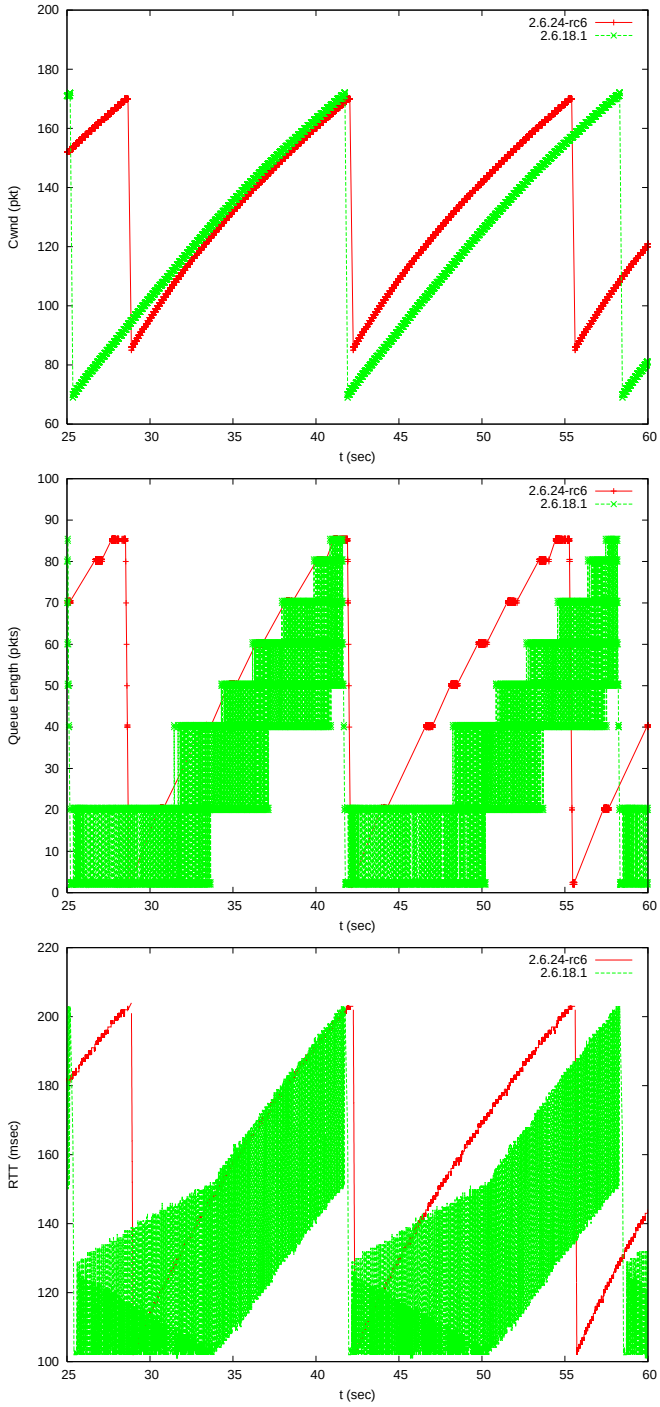
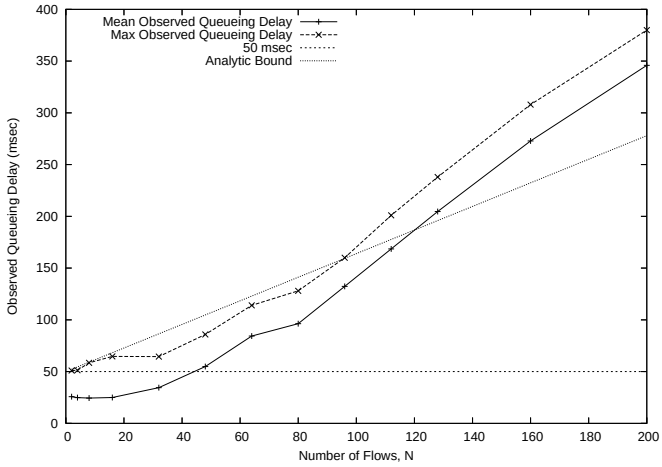
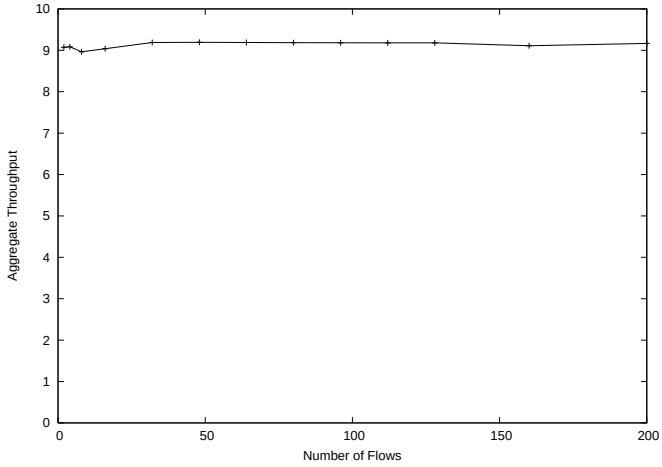


Fig. 13. Example time histories of measured RTT with TSO enabled in Linux 2.6.18 and 2.6.24 kernels. Delayed acking disabled. Top plot shows flow cwnd, middle plot queue occupancy and lower plot measured RTT at sender. 10Mbps link, single flow, RTT 100ms, queue 1×BDP.



(a) Peak and mean queueing delay vs number of flows



(b) Goodput vs number of flows

Fig. 14. Performance with delayed acking and TCP segmentation offload enabled. 10Mbps link. Flow RTTs uniformly random from 20-200ms.

VI. Scope

This paper focusses on the specific question of whether low correlation between measured and network congestion is a fundamental obstacle to congestion control. This issue is of particular interest as it is a commonly voiced concern and has been the subject of a number of published studies. We emphasise that many other issues that are not addressed here remain to be considered before any decision could be made as to the suitability or otherwise of delay as a congestion signal for use other than on a purely experimental basis. For example, we do not consider the issue of co-existence between flows operating loss-based and delay-based congestion control, performance over multiple bottlenecks, performance over wireless links and so on. Nevertheless, we argue that the work here is an important first step in exploring the nature of fundamental constraints on congestion control.

VII. Conclusions

In this paper we revisit the commonly voiced concern that low correlation between measured delay and network congestion that delay may be fundamentally flawed as a signal

for congestion control. This concern is particularly topical in view of a number of proposals to change the TCP congestion control algorithm to make use of delay information. Examples include not only FAST TCP [7] but also hybrid congestion control algorithms based on the use of both loss and delay, e.g. TCP Illinois [8] and Compound TCP [15]. The latter is now available in Windows Vista and is currently undergoing review at the IRTF and IETF standards bodies.

Our main contribution in this paper is to demonstrate that in fact what matters for congestion control is the *aggregate* behaviour of the flows sharing a link. Hence, perhaps somewhat surprisingly, while any given single flow may measure delay which is only weakly correlated with network congestion, this is not in itself an obstacle to congestion control. In this paper we demonstrate this constructively via detailed experimental tests and also confirm analytically the general nature of our conclusions. This potentially has direct implications for our understanding of the fundamental constraints on congestion control within the current Internet architecture.

ACKNOWLEDGEMENT

Discussions with Robert Shorten are gratefully acknowledged.

REFERENCES

[1] S. Biaz and N. Vaidya. Is the round-trip time correlated with the number of packets in flight? In *Proceedings Internet Measurement Conference (IMC)*, 2003.

[2] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of SIGCOMM*, pages 24–35, 1994.

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

[4] D.J.leith, J.Heffner, R.N.Shorten, and G.D.McCullagh. Delay-based aimd congestion control. In *Proc. Workshop on Protocols for Fast Long Distance Networks, Los Angeles.*, 2007.

[5] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. IETF RFC 1323, 1992.

[6] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM Computer Communication Review*, 19(5):56–71, 1989.

[7] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, 2004.

[8] S. Liu, T. Ba?ar, and R. Srikant. TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks. In *Proc. First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), Pisa, Italy, October 11-13, 2006*, 2006.

[9] J. Martin, A. Nilsson, and I. Rhee. Delay-based congestion avoidance for TCP. *IEEE/ACM Transactions on Networking*, 11(3):356–369, June 2003.

[10] R. S. Prasad, M. Jain, and C. Dovrolis. On the effectiveness of delay-based congestion avoidance. In *Second International Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[11] S. Rewaskar, J. Kaur, and D. Smith. Why don't delay-based congestion estimators work in the real-world? Technical Report TR06-001, Department of Computer Science, UNC Chapel Hill, July 2005, 2005.

[12] R.N.Shorten, D.J.Leith, and F.Wirth. Products of random matrices and the internet: Asymptotic results. *IEEE Transactions on Networking, 14(6), pp. 616-629*, 2006.

[13] R.N.Shorten, D.J.Leith, J.Foy, and R.Kilduff. Analysis and design of congestion control in synchronised communication networks. Automatica, 2004.

[14] R. Shorten and D. Leith. On queue provisioning, network efficiency and the delay-bandwidth product. *IEEE Transactions on Networking*, 15:866–877, 2007.

[15] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *International Workshop on Protocols for Fast Long-Distance Networks*, 2005.

[16] Z. Wang and J. Crowcroft. A new congestion control scheme: Slow Start and Search (Tri-S). *ACM Computer Communication Review*, 21(1):32–43, 1991.

[17] Z. Wang and J. Crowcroft. Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm. *ACM Computer Communication Review*, 22(2):9–16, 1992.

Doug Leith Doug Leith graduated from the University of Glasgow in 1986 and was awarded his PhD, also from the University of Glasgow, in 1989. In 2001, Prof. Leith moved to the National University of Ireland, Maynooth to assume the position of SFI Principal Investigator and to establish the Hamilton Institute (www.hamilton.ie) of which he is Director. His current research interests include the analysis and design of network congestion control and distributed resource allocation in wireless networks.

Gavin McCullagh