On queue provisioning, network efficiency and the Transmission Control Protocol

R. N. Shorten, D.J.Leith,

Hamilton Institute, NUI Maynooth, Ireland.

$\{robert.shorten,\ doug.leith\}@nuim.ie$

Abstract— **In this paper we propose a sender side modification to TCP to accommodate small network buffers. We exploit the fact that the manner in which network buffers are provisioned is intimately related to the manner in which TCP operates. However, rather than designing buffers to accommodate the TCP AIMD algorithm, as is the traditional approach in network design, we suggest simple modifications to the AIMD algorithm to accommodate buffers of any size in the network. We demonstrate that networks with small buffers can be designed that transport TCP traffic in an efficient manner while retaining fairness and friendliness with standard TCP traffic.**

I. INTRODUCTION

A key issue in the design of internet routers is that of buffer sizing. Router buffers are usually sized with two primary objectives in mind.

(i) *Accommodating short-term packet bursts.* Due to the nature of transport protocols such as TCP, internet traffic tends to be bursty. Should too many packets arrive in a sufficiently short interval of time then the egress link lacks the capacity to process all of the packets immediately. The first job of the router buffer is to mitigate packet loss due to bursts by accommodating these packets in a queue until they can be serviced.

(ii) *Ensuring AIMD throughput efficiency.* Most network traffic is carried by the TCP. The AIMD congestion control algorithm used by TCP reduces the number of packets in flight by half on detecting network congestion. If network queues are too small, this backoff action will cause them to empty with a corresponding reduction in link utilisation.

Router buffers are designed with both of these objectives in mind; the buffer size should be large enough to accommodate typical packet bursts in the network, and should be chosen to so that the buffer does not empty for significant periods of time when TCP responds to network congestion. The typical rule of thumb in the design of router buffers is to provision the buffer to be equal to the bandwidth of the link served by the router (measured in packets per second) multiplied by the average round trip time of the flows utilising the router (RTT_{av}.): the *Delay-Bandwidth Product* (DBP). While provisioning network buffers in this manner has served the networking community well in the past, it is generally accepted that buffers in future routers are unlikely to be provisioned in this manner. For example, strong arguments are given in [1] to suggest that

queues provisioned according to the DBP rule may result in unacceptable large absolute variations in the queueing delay in high delay-bandwidth networks, and may be very difficult to realise physically.

An alternative strategy to the DBP rule for buffer provisioning is to exploit statistical multiplexing effects of packets arriving at network buffers to justify arguments in favour of smaller buffer sizes; see the recent papers [1], [?], [?] and the references therein for a summary of work in this direction. Roughly speaking, these papers all suggest that one may expoloit statistical multiplexing of TCP flows to enable deployment of much smaller router buffers than that suggested by the DBP rule (without adversely affecting link utilisation). While approaches of this type are of merit, and certainly provide key insights into the behaviour of networks, they are crucially dependent upon the assumptions that (i) the buffer of interest serves a large number of flows at any instant of time and (ii) only a small proportion of flows perform an AIMD backoff in response to network congestion (i.e. flows are not synchronised). If these assumptions do not hold then provisioning network buffers in this manner will lead to poor utilisation of the bottleneck link bandwidth. This is illustrated, for example, in Figure 1 which shows link utilisation versus buffer size for a single TCP flow and for 100 TCP flows. With a single flow, when congestion occurs the flow reduces the number of packets in flight by half. When the queue size is small, this leads to the queue emptying for a significant period of time before the probing action of the AIMD congestion control leads to it filling again. Hence, link utilisation is lowered. With 100 flows, at any given congestion event on average only a relatively small proportion of flows will backoff and hence the likelihood of the queue completely emptying is less. This statistical multiplexing of flow backoffs means that link utilisation is on average higher.

The example in Figure 1 also indirectly highlights a further fundamental issue. In this example all flows are long-lived and the number of flows is constant over the life of each experiment, but in practice a network can be expected to contain flows with a broad mix of connection sizes and where flows frequently start/stop. In such a rapidly changing packet-switched environment, it is far from straightforward to define, or to measure, the "number of flows" at any instant. Further, the traffic mix can be expected to change significantly over time e.g. over the course of a day. It is unclear how buffer sizing rules based on the number of long-lived flows might be applied in such environments.
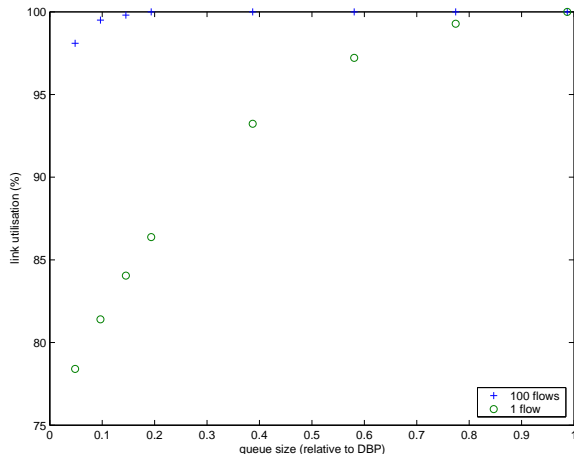
Fig. 1. Link utilisation vs buffer size and number of TCP flows (NS simulation: bandwidth 100Mb, average delay 80ms).

These observations suggest that one must either dynamically adjust the router buffer size to regulate utilisation (as suggested in [?]) or adjust the end-to-end protocols themselves so as to ensure high network utilisation (irrespective of the level of flow synchronisation, or the number of flows traversing the link). In this paper we explore the latter of these alternatives. We suggest modifications to the TCP AIMD algorithm to address the buffer-provisioning problem in a way that retains the benefits of statistical multiplexing when many flows share a link while also achieving good link utilisation with small numbers of flows. We exploit the fact that the manner in which network buffers are provisioned is intimately related to the manner in which TCP operates. However, rather than designing buffers to accommodate the TCP AIMD algorithm, we suggest simple modifications to the AIMD algorithm to accommodate buffers in the network. We shall see that with only minor modifications to the AIMD algorithm, networks with small buffers can be designed that transport TCP traffic in an efficient manner.

Our paper is structured as follows. We begin the discussion by reviewing the origins of the DBP rule and by discussing the relationship between queue provisioning and throughput through the link served by the buffer. We then revisit the DBP rule and suggest an alternative strategy to maintain high link utilisation through the bottleneck link. As discussed above our proposal involves suggesting minor modifications to the standard TCP algorithm and we discuss the relevant implementation issues, including presenting a number of network measurements that demonstrate the efficacy of the proposed algorithm in real networks. We also examine the effect of these modifications on the fairness, friendliness and convergence rate of networks carrying TCP traffic. Finally, our conclusions are summarised in Section 8.

II. THE DELAY-BANDWIDTH PRODUCT RULE

AIMD congestion control operates a window based congestion control strategy. Source i maintains an internal variable $cwnd_i$ (the congestion window size) which tracks the number of sent unacknowledged packets that can be in transit at any time, i.e. the number of packets in flight. On safe receipt of data packets the destination sends acknowledgement (ACK) packets to inform the source. When the window size is exhausted, the source must wait for an ACK before sending a new packet. Congestion control is achieved by dynamically adapting the window size according to an additive-increase multiplicative-decrease (AIMD) law. Roughly speaking, the basic idea is for a source to probe the network for spare capacity by increasing the rate at which packets are inserted into the network, and to rapidly decrease the number of packets transmitted through the network when congestion is detected through the loss of data packets. In more detail, the source increments $cwnd_i(t)$ by a fixed amount $\alpha_i/cwnd_i(t)$ upon receipt of each ACK. On detecting packet loss, the variable $cwnd_i(t)$ is reduced in multiplicative fashion to $\beta_i cwnd_i(t)$. Standard TCP uses the values $\alpha_i = 1$ and $\beta_i = 0.5$.

While the basic function of congestion control is to regulate network congestion, it is clearly desirable to also ensure that the network flows, on aggregate, fully utilise the available network resources. Consider, initially, a network with a single bottleneck link (multiple bottlenecks will be considered later). When the network experiences congestion the link buffer is full and the network bottleneck is necessarily operating at link capacity. The corresponding data throughput through the bottleneck link is given by

$$R(k)^- \quad = \quad \sum_{i=1}^{n} \frac{w_i(k)}{T_i + \frac{q_{max}}{B}} = B \qquad (1)$$

where k indexes the instant just before sources respond to the k'th network congestion event and w_i denotes the value of $cwnd_i$ when congestion is detected by each source. B is the link capacity, q_{max} is the bottleneck buffer size, T_i is the round-trip-time experienced by the $i'th$ source when the bottleneck queue is empty and $T_i + q_{max}/B$ is the round-trip time when the queue is full. Here $R(k)^-$ denotes an event just before packets are dropped due to overflow of the buffer, and $R(k)^+$ after each source has responded to the k'th congestion event by reducing its number of packets in flight.

We let $\beta_i(k) = \beta_i$ if flow i experiences a loss at the k'th congestion event and otherwise $\beta_i(k) = 1$ (i.e. flow i does not backoff). Then, following congestion, the data throughput is given by

$$R(k)^+ \quad = \quad \sum_{i=1}^{n} \frac{\beta_i(k) w_i(k)}{T_i} \qquad (2)$$

under the assumption that the bottleneck buffer empties[1]. If the sources backoff too much, data throughput will suffer as the queue will empty for a period of time and thus the link will operate below its maximum rate. A simple method to ensure maximum throughput is to equate the rates $R(k)^-$ and $R(k)^+$. This can be achieved by enforcing the following constraint,

$$\beta_i \geq \frac{T_i}{T_i + \frac{q_{max}}{B}} = \frac{RTT_{min,i}}{RTT_{max,i}}. \qquad (3)$$

For a given choice of β_i one may seek to choose q_{max} such that $R(k)^+ = R(k)^-$ for all k. Evidently, for the case of

[1]This assumption merely streamlines the presentation. If the queue does not empty at the k'th congestion event, we have trivially that $R(k)^+ = R(k)^-$ and link utilisation is 100%.

networks employing standard TCP, namely $\beta_i = 0.5$, it follows $q_{max} \geq BT_{d_i}$. This is the origin of the DBP rule.

A. The DBP Rule & Statistical Multiplexing

The DBP rule is derived from consideration of link utilisation in the worst case where all flows backoff at each congestion event. While this situation applies when only a single TCP flow is present, it also also occurs with many flows provided that the flows are synchronised i.e. every flow experiences a loss at every congestion event. When flows are not synchronised, by definition only a proportion of flows backoff at each congestion event. Thus on average more packets remain in flight than in the synchronised case and so the queue is less likely to empty. Consequently, under unsynchronised conditions we expect that the link utilisation will be strictly greater than under synchronisation (assuming the queue is sized less than the BDP - otherwise the utilisation is trivially always 100%). This behaviour can be seen, for example, in Figure 3 which shows the link utilisation achieved by two TCP flows as the round-trip time of the second flow is varied. While the link utilisation varies in a complex manner as the flows move in and out of synchronisation (this type of behaviour is well known and associated with so-called phase effects), it can be seen that the utilisation always respects the synchronised case lower bound on efficiency that is marked on the figure.
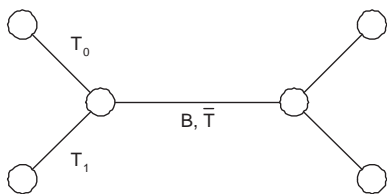
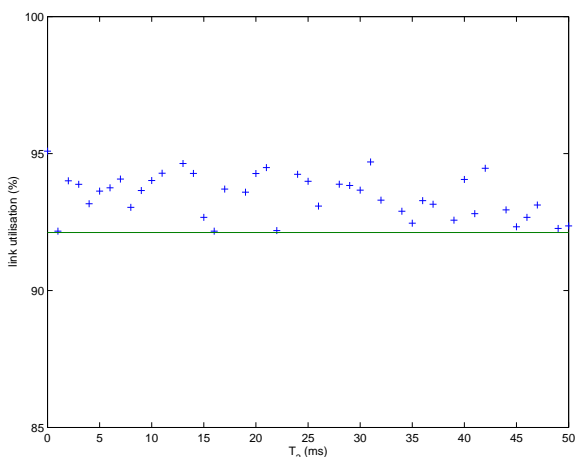

Fig. 2.   Dumbbell topology.

Fig. 3.   Link utilisation vs difference in round-trip times for two TCP flows and a dumbbell topology. Solid line marks link utilisation with drop synchronisation. (NS simulation, $\bar{T} = 50ms$, T_1=1.2ms, B=10Mbs, queue size scaled with T_2 to maintain a constant provisioning ratio of 0.75).

Appenzeller *et al* [1] and others have observed that synchronisation becomes rare as the number of competing TCP flows

increases. Thus, when there are many flows the opportunity exists to use smaller queues with little loss in efficiency. This behaviour is illustrated, for example, in Figure 4. In this figure the minimum queue size to ensure 99.99% link utilisation is plotted as the number of competing TCP flows is increased. The ∘ markers denote results when the TCP flows all have the same round-trip time and the delay-bandwidth product is 85 packets. It can be seen that for up to 45 flows it is necessary to size the queue at the delay-bandwidth product owing to the presence of synchronisation. However, for larger numbers of flows the queue size required rapidly falls to around 50 packets. Note that in this example the provisioning requirement does not fall below about 50 packets as the number of flows is increased further. Figure 4 also illustrates the corresponding behaviour when the flows have different round-trip times. We can see that in this case large queues are still necessary for small numbers of flows but that the queue provisioning requirement quickly decreases and, indeed, falls substantially below the 50 packet limit that was evident in the same round-trip time case.
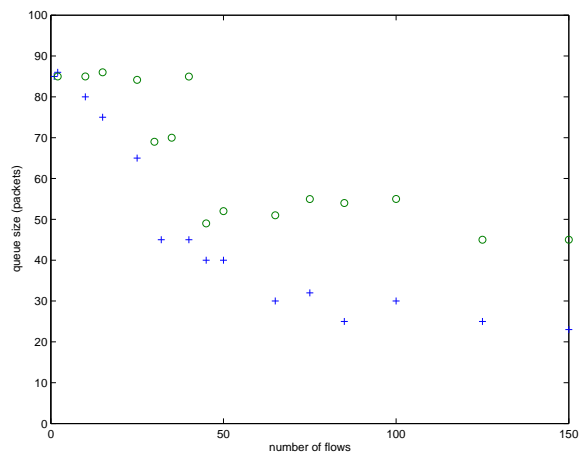

Fig. 4.   Minimum queue size for 99.99% link utilisation vs number of TCP flows. Dumbbell topology. Key: ∘ flows have the same round-trip time (T_2=1.2ms); + flows have different round-trip times (T_2 uniformly distributed in interval [0,30ms]). (NS simulation, $\bar{T} = 50ms$, T_1=1.2ms, B=10Mbs).

In general, we expect that the actual reduction in queue size that can be achieved while maintaining a required link utilisation depends in a complex manner on details of the network and prevailing traffic conditions. Nevertheless, under some simplifying assumptions (flow backoff's are independent, individual flow congestion windows are uniformly distributed about the same average size and there are sufficiently many flows that the aggregate is normally distributed) Appenzeller *et al* [1] shows that the queue size can be scaled as $1/\sqrt{n}$ and the utility of this result is confirmed by considerable empirical evidence.

Note however that this reduction in queue size is crucially dependent on (i) the presence of many flows and (ii) absence of synchronisation. With few flows or when synchronisation is prevalent, high link utilisation continues to require large buffers provisioned in line with the DBP rule.

III. REVISITING THE DELAY-BANDWIDTH PRODUCT RULE: ADAPTIVE AIMD

The discussion in Section II illustrates a key property of networks that are designed to carry TCP traffic: namely, that queue provisioning is strongly coupled to the parameters of the TCP AIMD algorithm. With reference to Equation (3), efficient link utilisation is achieved when $R(k)^+ = R(k)^-$ after each congestion event. As already discussed, one strategy to achieve this goal is the DBP rule. The DBP rule is, however, only one of many strategies that could be adopted to ensure that Equation (3) is satisfied. In particular, for any given queue size q_{max} one may simply set

$$\beta_i = \frac{RTT_{min,i}}{RTT_{max,i}} \qquad (4)$$

for all i; thereby ensuring that $R(k)^+ = R(k)^-$ for all k. The effect of this AIMD modification can be seen in Figure 5. In this example the queue provisioning is less than the delay-bandwidth product and the queue empties for a substantial period following backoff by a factor of 0.5 (see the first backoff event in the figure) with an associated reduction in link utilisation. Once the flow adjusts its backoff factor to the level of buffer provisioning we can see that the queue now just empties following a backoff event and the link continues to operate at capacity as desired. With this approach, network queues are provisioned to accommodate the level of packet burstiness and to meet latency and jitter requirements, rather than to accommodate the TCP AIMD parameters.
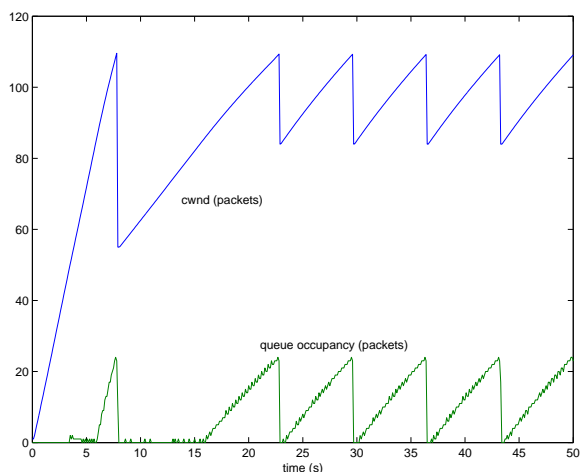

Fig. 5. Congestion window and queue occupancy time histories with adaptive AIMD algorithm. The delay bandwidth product is 85 packets. (NS simulation, bandwidth 10Mbs, RTT 100ms, queue 25 packets)

Adjustment of the flow backoff factors β_i can lead to unfairness between competing flows. However, fairness can be readily restored by making a corresponding adjustment of the flow increase parameters α_i. In a network of competing TCP flows, in equilibrium the mean peak congestion window of flow i is proportional to $\frac{\alpha_i}{\lambda_i(1-\beta_i)RTT_{min,i}}$ [?], where λ_i denotes the proportion of network congestion events at which flow i experiences a packet drop (in the synchronised case

$\lambda_i = 1$ for all flows)[2]. AIMD flows with increase and decrease parameters chosen such that the ratio $\alpha_i/(1-\beta_i)$ is the same will be fair if they have the same round-trip time and the same probability λ_i of experiencing a loss at congestion, see, for example, Figure 6. Hence, we adjust α_i according to

$$\alpha_i = 2(1-\beta_i) \qquad (5)$$

so that α_i is decreased as β_i increases thereby maintaining $\alpha_i/(1-\beta_i) = 2$. A ratio of 2 is selected as this corresponds to the ratio for standard TCP, where $\alpha_i = 1$, $\beta_i = 0.5$. In this way we maintain backward compatibility and friendliness towards legacy TCP flows (backward compatibility and support for incremental rollout are discussed in detail later).
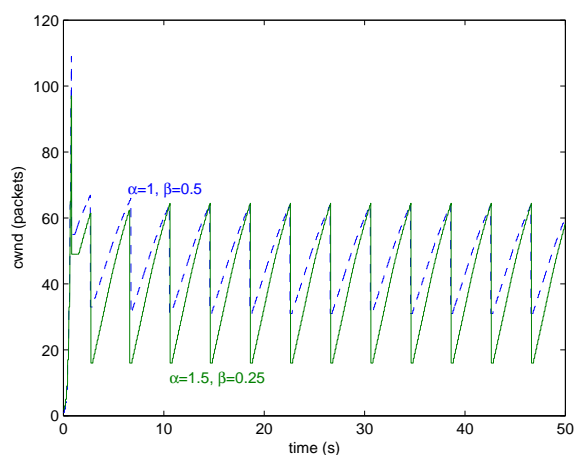

Fig. 6. Example of window fairness between two TCP sources with different increase and decrease parameters (NS simulation, network parameters: 10Mb bottleneck link, 100ms delay, queue 40 packets.)

Comment: Different RTTs. It is interesting to note that the proposed scheme elegantly deals with the situation where competing flows have different round-trip times. In this situation, with standard TCP for 100% utilisation the DBP rule requires the buffer to be sized according to the largest round-trip time. Of course, it is probably preferable to use instead an average round-trip time in order to avoid excessive queueing delay for those flows with short round-trip times, albeit at the price of reduced link utilisation. In contrast, with the adaptive scheme the buffer size is selected independently of the round-trip times of the flows and decision making is instead located at the network edges where each flow is individually responsible for adjusting its own backoff factor appropriately. In this way we avoid the trade-off between high link utilisation for flows with long round-trip times and low queueing latency for flows with short round-trip times.

Comment: Constraining β variations. Increasing the backoff factor β to improve efficiency decreases responsiveness (as measured by number of congestion epochs before convergence). Conversely, decreasing the backoff factor increases responsiveness but decreases efficiency, at least when queues are small. This issue is discussed in

[2]Note that λ_i here is quite different from the loss event probability p that is used in the Padye square root formula [2].

detail in Section **??** below. Note that large queues resolve the tradeoff between efficiency and responsiveness in favour of using a small backoff factor. However, when queues are small the situation is less clear. One approach is to restrict the backoff factor to an interval $[0.5, \beta_{max}]$. The value selected for β_{max} then reflects the preferred compromise between efficiency and responsiveness. We suggest that a reasonable choice for β_{max} in current networks might be 0.8 for the following two reasons: (i) a backoff factor of 0.8 ensures 100% link utilisation with queues sized at only 25% of the delay-bandwidth product which is already a fairly large reduction in the buffering requirement, and (ii) a backoff factor of 0.8 corresponds to a convergence time of 7 congestion epochs for 80% convergence and 12 congestion epochs for 95% convergence and so the slow down in responsiveness is relatively small. While this value is used in the rest of this paper, other choices of backoff limit are certainly possible and the final choice is left as future work.

Comment: Implementation. We note that the feasibility of obtaining reliable measurements of instantaneous RTT is an open question and do not propose use of this quantity. Minimum and maximum RTT are, however, relatively straightforward to estimate and this is discussed in more detail later. We note also that the proposed adaptive AIMD approach remains entirely within the well studied AIMD paradigm: the only change proposed is to the AIMD parameters used.

IV. Adaptive AIMD Performance

A. Single Bottleneck

We begin by considering the case of a single network bottleneck. By design, the adaptive AIMD scheme ensures full link utilisation across a wide range of queue provisioning levels (see Figure 7). In particular, for 100% utilisation under synchronised operation we require the queue size to be at least 25% of the delay-bandwidth product when the backoff factor is constrained to be less than $\beta_{max} = 0.8$.

As noted previously, this is a worst case bound on queue size. When flows are unsynchronised, the opportunity exists to use smaller queues with little loss in efficiency. This is illustrated, for example, in Figure 8. In this figure the queue sizes for 98% and 99.5% link utilisation are plotted as a function of number of flows for both the standard TCP AIMD algorithm and the adaptive AIMD algorithm proposed here. The results are for a network topology with a single bottleneck link and round-trip times distributed in the range 80-250ms. It can be seen that there is a substantial reduction in the queue provisioning requirement when the adaptive AIMD algorithm is used. Also evident is the approximate $1/\sqrt{n}$ dependence of queue size noted by [1]. We have obtained similar results over a wide range of network conditions although space restrictions mean that we cannot include them here.

The foregoing results relate to link utilisation. Figure 9 illustrates the impact of the adaptive algorithm on the network loss rate. This figure plots loss rate as the proportion of standard to adaptive TCP flows is varied. It can be seen that
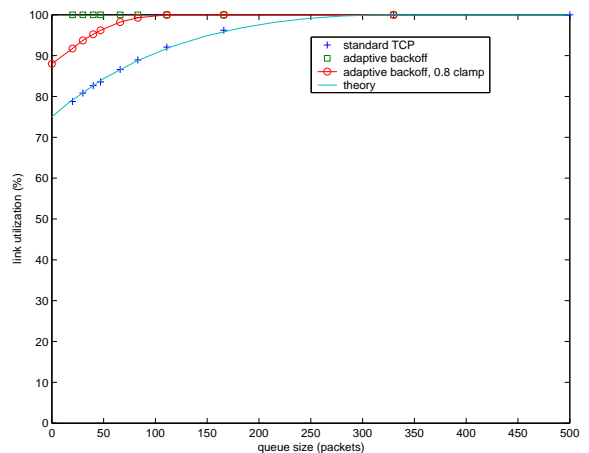

Fig. 7. Efficiency of TCP vs queue size. Results shown are for a single TCP flow, or multiple synchronised flows. The solid line is the theoretical efficiency curve given by Equation (**??**) in the Appendix. (NS simulation: bandwidth 100Mb, RTT 40ms).
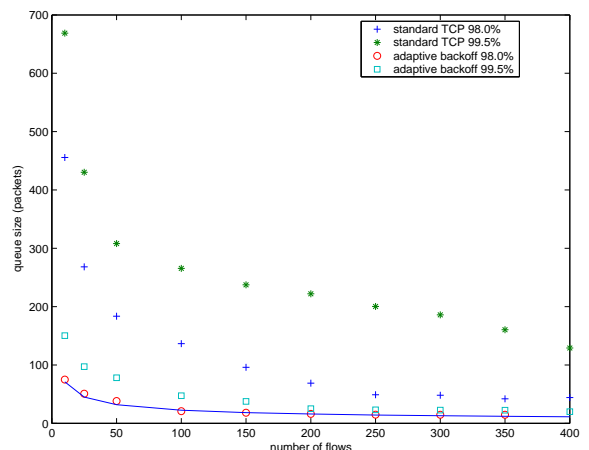

Fig. 8. Link utilisation vs queue size and number of flows. (bandwidth 155Mbs, delay 80-250ms). The solid line indicates a $1/\sqrt{n}$ curve corresponding to the 98% case with adaptive backoff.

the loss rate, measured as the proportion of sent packets that are dropped, falls as the number of adaptive flows is increased. This can be understood by noting that the adaptive algorithm adjusts the flow increase and decrease parameters in a co-ordinated manner that maintains the same congestion epoch duration, and thus number of packet losses, as standard TCP when flows are synchronised. The increased throughput with the adaptive algorithm means, however, that the *proportion* of packets lost falls compared with standard TCP. Thus, the adaptive algorithm protocol serves not only to increase link utilisation, but also the network goodput as well.

B. Mix of Connection Lengths

Real network paths typically contain TCP flows with a wide mix of connection sizes. We can expect this to reduce the benefits of adaptive AIMD as flows that complete before exiting slow-start gain no benefit from adapting their AIMD backoff factor. Further, it is possible that the presence of many flows in slow-start may increase the burstiness of traffic
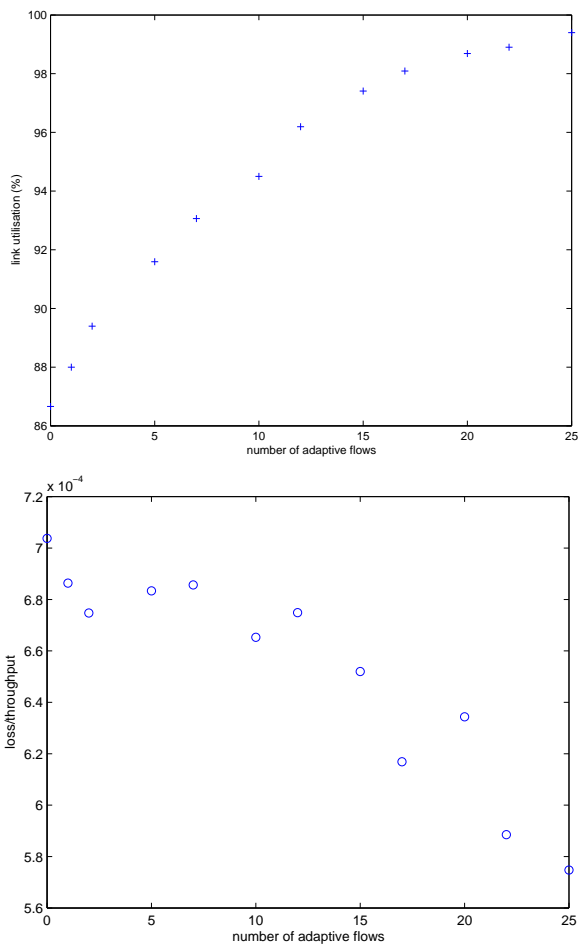
Fig. 9. Utilisation and loss rate as function of number of adaptive and conventional TCP flows (out of 25 flows in total, remaining flows are conventional TCP flows). NS simulation, 155Mb bottleneck link, 40ms delay, queue 25 packets.

Fig. 10. Link utilisation vs queue size and number of flows for a mix of long-lived flows and web traffic. (bandwidth 155Mbs, delay 80-250ms).

C. Impact of Reverse Path Traffic

It is well known that reverse-path traffic can increase the burstiness of the forward path TCP stream as a result of ACK compression and ACK losses. Increased burstiness typically increases the likelihood of packet losses when small queues are used, and may thus constrain the minimum queue size that can be used. We note, however, that the cost of a back-off event due to a reverse-path induced network burst is dependent on the amount by which a flow releases bandwidth after a loss event. In the presence of small buffers, the adaptive AIMD algorithm acts to decrease aggressiveness and to increase the back-off factors of network flows. Thus, we expect to see an improvement in network performance, compared to standard TCP, on links with small queues and reverse path traffic.

We present two sets of experimental results to illustrate the performance of the proposed adaptive algorithm in networks in which there is reverse path queuing, see Figures 11-12. These plots show link utilisation and loss rate as the proportion of standard to adaptive TCP flows is varied. Figure 11 shows results for 10 long-lived reverse path flows, while Figure 12 shows the corresponding results with 20 long-lived reverse path flows. We note that long-lived, rather than short-lived, reverse path traffic is the most demanding case as it creates sustained queueing on the reverse path, leading to persistent ACK compression and substantial ACK losses for the forward path flows. It can be seen that the adaptive AIMD flows achieve both significantly improved link utilisation (e.g. increasing utilisation from 55% to 79% with 20 reverse flows) and reduced loss rate (by approximately 50%) compared to standard TCP flows.

D. Multiple bottlenecks

It may occur that packets are queued at multiple queues, e.g. at the ingress and egress access links along a path or, as considered in the previous section, due to reverse path traffic. Considering initially the worst case (from a link utilisation viewpoint) situation where flows are synchronised, it is easy to see that the proposed adaptive backoff strategy can be readily

arrivals. Recall that queue sizing is determined both by the AIMD backoff characteristics and by the level of burstiness. Figure 10 plots the queue size required for 98.0% and 99.5% utilisation on a 155Mbs link with a mix of round-trip times and a mix of long-lived flows and web traffic. Web traffic connection sizes are generated according to a Pareto distribution, with exponentially distributed inter-arrival times between web sessions. Web traffic is bi-directional and the overall number of web sessions is chosen such that the mean number of web sessions active at any time is approximately equal to the number of long-lived connections. Comparing Figure 10 with Figure 8, we can see that the trends are similar; that is, that the required buffer size tends to decrease as the number of flows is increased and that the adaptive backoff scheme supports significant reductions in buffer size, compared with standard TCP, particularly when there are smaller numbers of flows. Observe that the overall size of buffer required is rather larger than that in Figure 8 when no web traffic is present. This is perhaps unsurprising – we attribute this difference to the presence of web flows persistently operating in slow-start mode resulting in both an increased level of packet burstiness and an increased packet loss rate for long-lived flows.
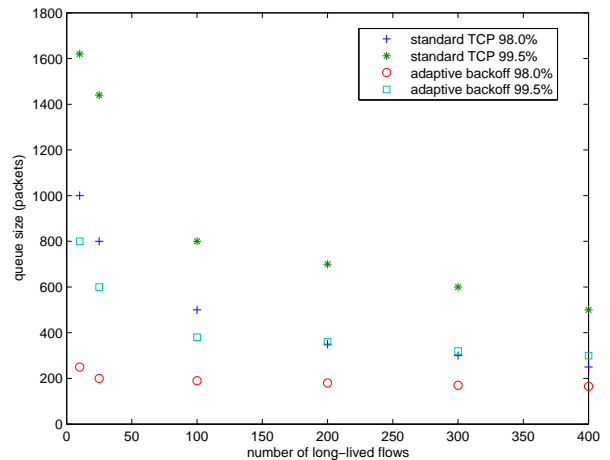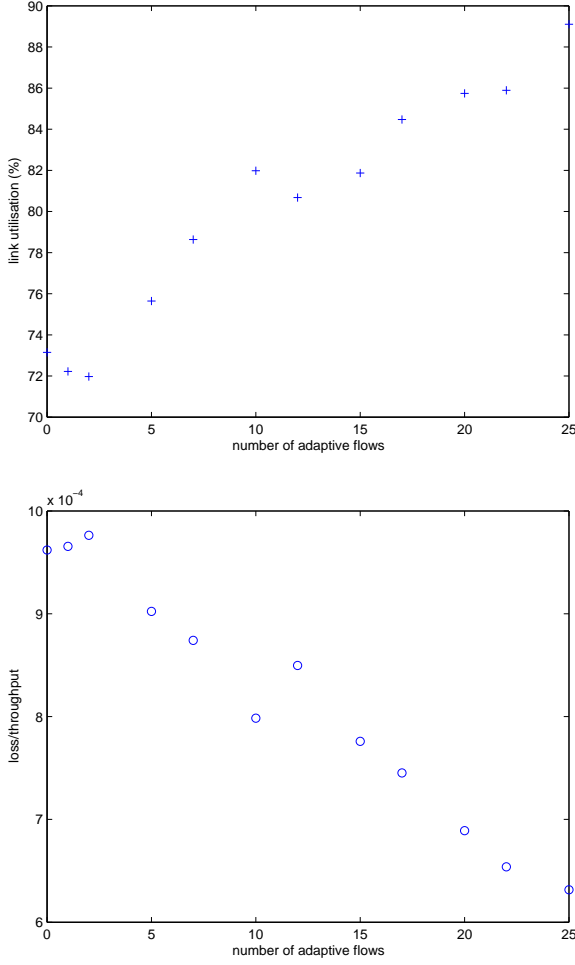
Fig. 11. Utilisation and loss as function of number of adaptive and conventional TCP flows (out of 25 flows in total, remaining flows are conventional TCP flows)with reverse path flows (10 long-lived TCP connections). NS simulation, 155Mb bottleneck link, 40ms delay, queue 125 packets.
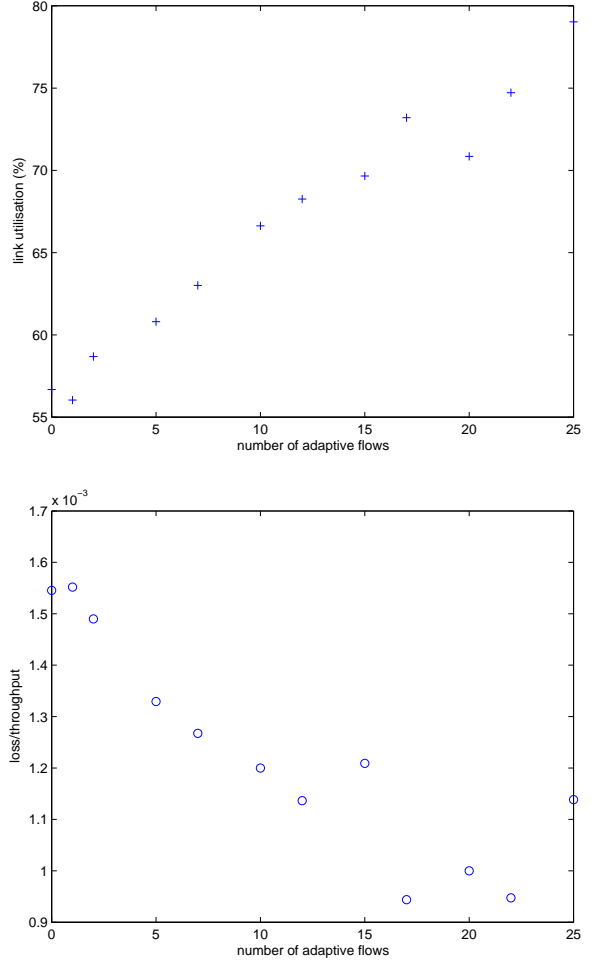
Fig. 12. Utilisation as function of number of adaptive and conventional TCP flows (out of 25 flows in total, remaining flows are conventional TCP flows)with reverse path flows (20 long-lived TCP connections). NS simulation, 155Mb bottleneck link, 40ms delay, queue 125 packets.

extended to ensure high link utilisation at the congested links in multiple bottleneck topologies. At congestion we have that

$$R(k)^- \quad = \quad \sum_{i=1}^{n} \frac{w_i(k)}{\sum_{j=1}^{m}(T_i + q_j(k)/B_j)} = B \qquad (6)$$

where m is the number of links at which packets are queued, $q_j(k)$ is the queue occupancy of the j such link and B_j the bandwidth. Selecting the backoff factor as

$$\beta_i(k) \quad = \quad \frac{T_i}{T_i + \sum_{j=1}^{m} q_j(k)/B_j} = \frac{RTT_{min,i}}{RTT_{max,i}} \qquad (7)$$

then after backoff,

$$R(k)^+ \quad = \quad \sum_{i}^{n} \frac{T_i}{T_i + \sum_{j=1}^{m} q_j(k)/B_j} \frac{w_i(k)}{T_i} = B \qquad (8)$$

That is, the TCP flows adapt their backoff factors to just empty all of the queues that they see along the end-to-end path.

Comment: Number of Bottlenecks. Note that this analysis encompasses situations where different flows may see different numbers of backlogged queues along their respective network paths.

Comment: Reverse-path Queueing. Note also that since RTT measures the two-way delay along a path it includes the effect of reverse path queueing. In terms of adjusting flow backoff factors, we consider reverse path queueing to be simply a multiple bottleneck situation. By using a value of RTT_{max} that reflects reverse path queueing, our algorithm will correctly adapt flow backoff factors to empty both the forward and reverse path queues.

Comment: Variation in Bottleneck Number. The analysis also extends to situations where, for example as flows start and stop, the number of bottleneck links may vary - see example below. To implement the backoff calculation in (7), we use $RTT_{min,i}/RTT_{max,i}$ as before. Increases in the number of bottleneck links generally lead to an increase in round-trip time and this will be immediately reflected in the value of $RTT_{max,i}$. To capture changes that lead to a reduction in the round-trip time, we add exponentially fading memory to our estimate of $RTT_{max,i}$, namely at each congestion event we

reset our $RTT_{max,i}$ estimate according to

$$RTT_{max,i} \quad = \quad RTT_{min,i} + a(RTT_{max,i} - RTT_{min,i}) \quad (9)$$

with $a < 1$. We have not found the performance of the algorithm to be especially sensitive to the value of a used and suggest that a reasonable value of $a = 7/8$, which corresponds to the fading memory already used in the smoothed RTT calculation in TCP.

For example, consider the two bottleneck topology shown in Figure 13. Flow 1's fair share of the 100Mbs link is approximately 50Mbs but owing to the 40Mbs link it cannot achieve this. Hence, when only flow 1 is active, there exists a single bottleneck with packets queueing at the 40Mbs hop. When flow 2 becomes active, flow 1 continues to be constrained to 40Mbs throughput (with packets queueing at the 40Mbs link) while flow 2 obtains 60Mbs bandwidth on the 100Mbs link (with flow 2 packets queueing at this link, along with flow 1 packets that are in transit). When a packet drop occurs, the round-trip time of flow 1 is $T_1 + q_1/B_1 + q_2/B_2$ while that of flow 2 is $T_2 + q_2/B_2$ and the flow backoff factors adapt to just empty all of the queues in the network, see Figure 14.
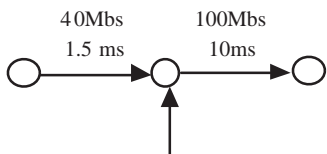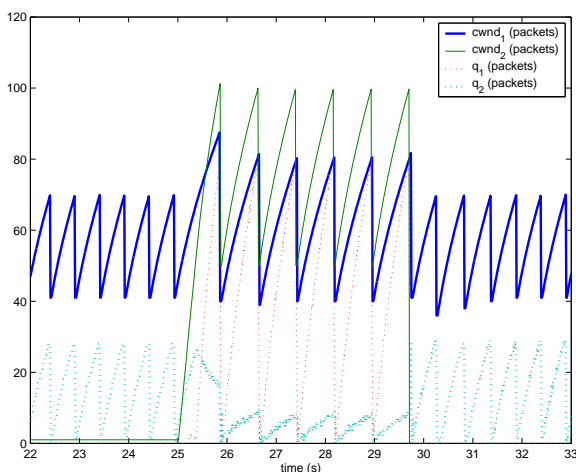

Fig. 13.   Two bottleneck topology.

Fig. 14.   Example of backoff adjustment with cross-flow from 25s-30s (NS simulation, 2 hop topology of Figure 13, hop 1: 40Mbs, 1.5ms delay, queue 30 packets, hop 2:100Mbs, 10ms delay, queue 80 packets.)

E. Support for incremental rollout

Incremental rollout requires (i) backward compatibility with legacy flows i.e. TCP friendliness, and (ii) that any proposed change to TCP yields worthwhile benefits without requiring requiring universal adoption or a "big bang" rollout. As noted previously, the proposed adaptive AIMD algorithm adjusts its

AIMD increase and decrease parameters in a co-ordinated manner that ensures backward compatibility with legacy TCP flows. This is illustrated in Figure 15, where in a network of 25 flows the proportion of standard and adaptive AIMD flows is varied. It can be seen that the ratio of the mean peak congestion windows of the standard and adaptive flows always stays close to unity. Figure 9 shows the corresponding impact on link utilisation. For the proposed adaptive AIMD algorithm, we can see that there is a strictly increasing gain in efficiency as the number of adaptive flows is increased. That is, there is an efficiency gain even if only a small percentage of flows utilise the adaptive algorithm.
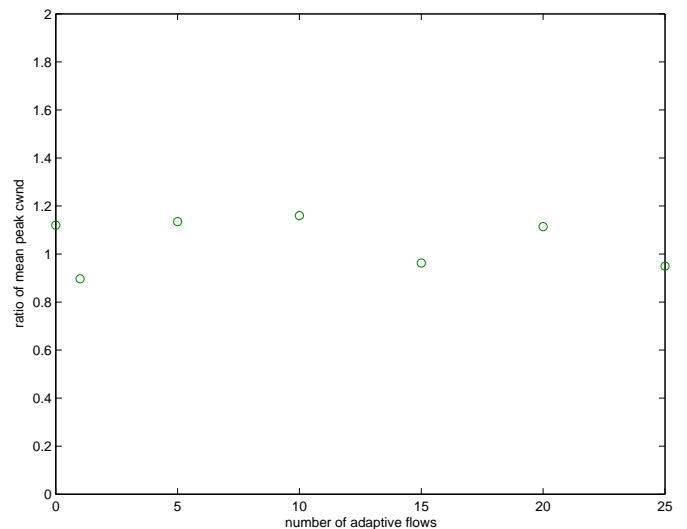

Fig. 15.   Fairness of adaptive and conventional TCP flows vs number of adaptive AIMD flows (out of 25 flows in total, remaining flows are conventional TCP flows). NS simulation, 155Mb bottleneck link, 120ms delay, queue 125 packets. Network includes background web traffic of approximately 1% link bandwidth.

F. Experimental results

In addition to NS-2 simulation tests, we have implemented the adaptive AIMD algorithm in Linux and evaluated its performance on an instrumented test-bed network. This network consisted of six Linux Xeon 2.8HGz servers with PCI-X Intel Pro 1000 NICs and a similarly specified router running FreeBSD 4.8 and using DummyNet to emulate specified network propagation delays. TCP traffic is generated using iperf. Table I presents a sample of the results we have obtained on this test-bed network for a range of network conditions, queue sizes and number of flows. The reported throughput values are averages taken over 10 minute test runs. The results obtained are in excellent agreement with the theoretical analysis and simulation results presented earlier.

We have also carried out a number of tests over production networks. The range of network conditions and paths available to us over production networks is less flexible than in the test-bed network and measurements are more limited - for example, we cannot alter router queue sizes or control/measure other traffic sources. We have carried out tests on paths between servers located at the Hamilton Institute (Maynooth, Ireland),

at CERN (Geneva, Switzerland) and Lawrence-Berkeley National Lab (San Francisco, U.S.). In these tests the bottleneck is a 20Mbs link located in Ireland. Results are shown in Table II for different numbers of flows. It can be seen that, in line with our analysis, with adaptive AIMD the aggregate throughput achieved is largely independent of the number of flows. For reference, the corresponding results for standard TCP are also presented. Throughput is consistently lower than with adaptive backoff owing to under-buffering at the bottleneck link.

Link	Queue Provisioning (% BDP)	No. of Flows	Link utilisation (%)	
			Standard TCP	With Adaptive Backoff
20Mbs, 150ms	25	1	86.35	96.20
	4	10	88.30	94.95
	2	25	73.90	83.30
	2	50	83.75	89.45
40Mbs, 75ms	25	1	86.30	96.30
	4	10	88.13	95.05
	2	25	67.80	77.00
	2	50	80.23	87.63
200Mbs, 25ms	15	1	85.26	95.58
	8	10	89.91	96.08
	6	25	90.13	96.49
	4	50	90.98	94.49

TABLE I

EXPERIMENTAL MEASUREMENTS OF LINK UTILISATION VERSUS NUMBER OF FLOWS FOR BOTH STANDARD TCP AND TCP WITH ADAPTIVE BACKOFF OVER A RANGE OF NETWORK CONDITIONS.

Path	No. Flows	Throughput (Mbs)	
		Standard TCP	With Adaptive Backoff
HI-CERN (delay 30ms)	1	15.0	19.2
	5	17.4	19.1
	10	18.1	19.2
	25	18.9	19.2
HI-LBL (delay 152ms)	1	15.0	18.5
	5	15.7	18.9
	10	16.8	19.2
	25	17.7	19.2

TABLE II

EXPERIMENTAL MEASUREMENTS OF FLOW THROUGHPUT VERSUS NUMBER OF FLOWS FOR BOTH STANDARD TCP AND TCP WITH ADAPTIVE BACKOFF OVER PATHS IN THE INTERNET.

V. IMPLEMENTATION

The proposed adaptive AIMD algorithm requires that each TCP flow estimate $RTT_{min,i}$ and $RTT_{max,i}$. While the feasibility of obtaining reliable measurements of instantaneous RTT remains an open question, minimum and maximum RTT are relatively straightforward to estimate and this is discussed in more detail in this section.

RTT_{min} is the speed-of-light round-trip propagation delay along the path of a flow. We base our estimate of RTT_{min} on observing the minimum time elapsed between transmission of a data packet and receipt of the corresponding acknowledgement. This estimate is largely unaffected by variable factors such as queueing delays and reverse path traffic as these

tend to increase the transit time for a packet and are therefore filtered out by taking the minimum. We have observed instead that the main estimation issues are associated with the impact of route changes. Route changes that *reduce* the speed-of-light delay will be correctly detected by observing the minimum time elapsed between transmission of a data packet and receipt of the corresponding acknowledgement. However, routing changes that *increase* the propagation delay will not be detected by this approach and so the value of RTT_{min} estimated may be smaller than the actual propagation delay. In this case the backoff factor used in the adaptive AIMD algorithm is too small, leading to a reduction in efficiency for that flow (at worst reverting to the conventional TCP backoff factor of 0.5). While this might be resolved by adapting the RTT_{min} estimator in a suitable manner, we argue that it is an unnecessary complication since substantial increases in speed-of-light delay during the lifetime of a flow seem fairly rare and therefore the overall potential for efficiency loss is minor.

RTT_{max} estimates the sum of the propagation delay and the maximum queueing delay along a path. The maximum of the instantaneous RTT (measured using packet time stamps in most modern implementations) frequently overestimates RTT_{max} owing to delays introduced by delayed acking and other short term increases in packet transit time (e.g. routing table updates *etc* may temporarily increase router forwarding delay). Spurious short term increases in round-trip time can be filtered out by using the maximum of the *smoothed* RTT instead of the instantaneous RTT. The smoothed round-trip time is already used within the TCP timeout algorithm and so is readily available. While seeking to filter out spurious short-term increases, we do want our measurement to reflect longer term changes in delay due, for example, to changes in the number of bottleneck links, routing changes and so on. Changes that lead to an increase in delay will be immediately reflected in the value of RTT_{max}. As discussed previously in Section IV-D, changes that lead to a reduction in delay can be tracked by adding a fading memory to the RTT_{max} estimate, see Equation (9).

Our ability to effectively estimate $RTT_{min,i}$ and $RTT_{max,i}$ in challenging network conditions is illustrated in Figure 16. Here, the flow of interest shares a link with bidirectional web traffic. The web traffic, which is stochastic and bursty in nature, creates complex variations in both forward and reverse path queueing delays. It can be seen that the adaptive backoff scheme nevertheless performs well, correctly adjusting the backoff factor and rejecting the spurious delay spikes generated by the web traffic. Further evidence of the practicality of estimating $RTT_{min,i}$ and $RTT_{max,i}$ is provided by the experimental tests discussed in the previous section. While limitations on the range of experiments that we are able to perform mean that we cannot *prove* correct performance under all conditions, the foregoing analysis and results do provide good support of correct operation in practical situations.

Comment: Smoothed RTT & Throughput. We note that smoothed RTT reflects the average delay experienced by a large number of packets. It is therefore closely related to the achieved throughput sustained over several round-trip times,

from which we can see that substantial changes in this quantity are rarely spurious. This also leads us to the possibility of adapting the backoff factor based on throughput rather than round-trip time measurements if so desired. For example, we have that the backoff factor can be expressed as

$$\beta_i(k+1) = \min_j \beta_i(j) \frac{B_i^-(j)}{B_i^+(j)} \qquad (10)$$

where $B_i^-(k)$ is the throughput of flow i immediately before the k'th congestion event, $B_i^+(k)$ the throughput of flow i immediately after the k'th congestion event. Both quantities are readily measured from packets ACK'ed over a round-trip time. This avoids the need to measure the ratio $RTT_{min,i}/RTT_{max,i}$ directly.
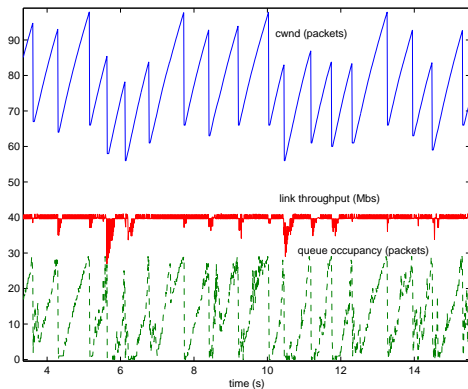

Fig. 16.   Example of operation with bidirectional web traffic sharing link (NS simulation, network parameters: 40Mb bottleneck link, 20ms delay, queue 30 packets, 10 web flows in each direction.)

VI. THE COST OF ADAPTIVE AIMD

It has been well documented that even small modifications to the basic TCP algorithm can have a large impact on network properties. In this section we consider the cost of our suggested modifications to the standard TCP algorithm. In particular, we consider the impact of our algorithm on the network convergence rate. From [?] we have that:

(i) **Convergence.** Consider a network of AIMD flows with drop-tail queues. For synchronised flows, the flow congestion windows converge to a unique periodic cycle at an exponential rate. In the case of unsynchronised flows, the mean peak congestion window of each flow converges exponentially to a unique equilibrium value.

(ii) **Convergence rate.** Convergence rate refers to the rate at which the mean congestion windows of the network flows converge to their equilibrium values, e.g. following start up of a new flow. In the case of synchronised flows, the convergence rate of the flow congestion windows is bounded by the largest backoff factor $\beta_{max} = max_i \beta_i$ in the network, with the 95% rise time measured in congestion epochs bounded by $\log 0.05 / \log \beta_{max}$ (yielding a rise time of 4 congestion epochs for a backoff factor of 0.5 and 10 congestion
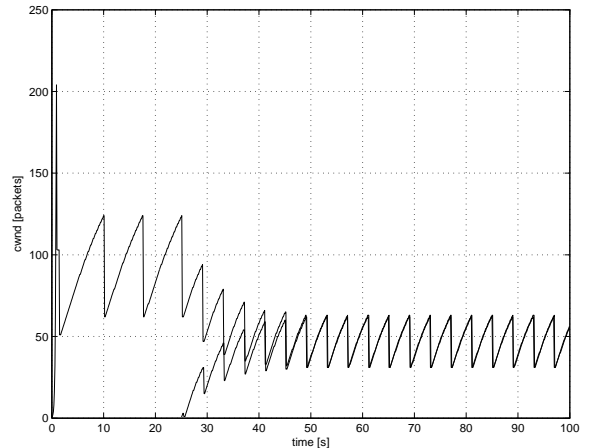

Fig. 17.   Illustration of TCP convergence rate (NS simulation, $\alpha_i = 1$, $\beta_i = 0.5$, dumb-bell with 10Mbs bottleneck bandwidth, 100ms propagation delay, 40 packet queue).

epochs for a backoff factor of 0.75). See, for example, Figure VI. When flows are unsynchronised, we work in terms of the *mean* backoff factor $E[\beta_i] = \lambda_i(1 - \beta_i)$. The convergence rate of the network *mean* congestion windows is bounded by the largest mean backoff factor $\beta_{max,E} = max_i E[\beta_i]$, with the 95% rise time bounded by $\log 0.05 / \log \beta_{max,E}$.

These analytic results indicate that congestion control strategies that reduce the AIMD backoff factors to achieve high utilisation of network resources can result in slowing of the rate of convergence of the network to its equilibrium. A backoff factor of 0.75 has a 95% rise time of 10 congestion epochs compared with a rise time of only 4 congestion epochs when the backoff factor is 0.5. For example, see Figure 18. Note that this analysis focusses on the congestion avoidance behaviour and ignores the impact of slow start which would tend to accelerate convergence following start-up of a new flow, although slow start would have little impact on the convergence time following a cross-flow disturbance. For the delay-bandwidth products commonly encountered in current networks, the impact on convergence time of increasing the backoff factor to 0.8 is felt to be minor. Hence, while a mode switch might be included to accelerate convergence, this does not seem necessary on low to medium delay-bandwidth product paths. In contrast, on high delay-bandwidth paths, such as transatlantic multi-gigabit speed paths, it is well known that the current AIMD algorithms suffer from slow convergence and our adaptive AIMD algorithm would obviously suffer similarly. A number of proposals exist that seek to improve performance in high delay-bandwidth product environments and, although these are outside the scope of the present paper, we do note that any proposed changes compatible with the existing TCP AIMD algorithm would also be compatible with the adaptive AIMD algorithm.

VII. SCOPE OF OUR RESULTS AND FUTURE WORK

The analysis and results we present here encompass a wide range of network conditions including, in particular, paths
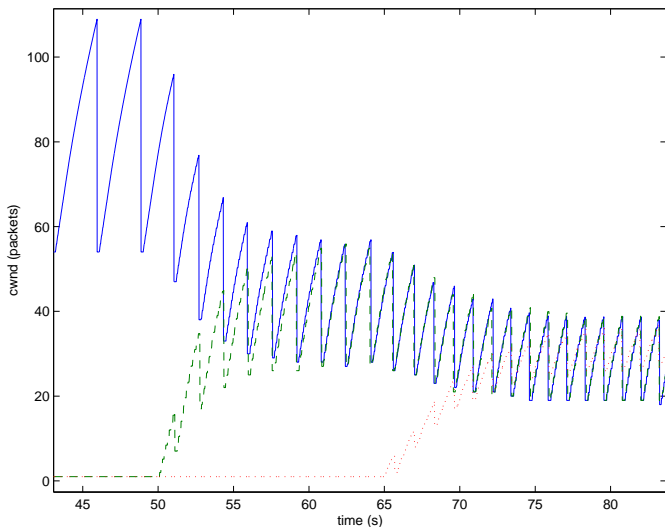
Fig. 18. Illustrating slower responsiveness with larger backoff factors: flow starting at 50s has backoff factor of 0.5, flow starting at 65s has backoff factor of 0.75. NS simulation, 10Mb bottleneck link, 80ms delay, queue size 40 packets.

with both single and multiple bottlenecks (the number of which may change over time), a mix of long-lived flows and web traffic, and competing flows with a range of different round-trip times. Issues which are not considered in this paper and are left as future work include link utilisation with active queueing disciplines (we justify our focus on drop-tail queueing by noting that it remains the prevalent queueing discipline in modern networks) and the impact of wireless links on utilisation.

VIII. RELATED WORK

One of the first mentions of the DBP rule seems to be in an RFC by Jacobson *et al* [3] although it is also implicit in the original Jacobson TCP paper [4]; subsequent simulation studies include the work by Villamizar and Song [5]. The role of statistical multiplexing in allowing small buffers is discussed in detail in the recent paper by Appenzeller *et al* [1] and the reader is referred to the references therein for details of other work on this topic.

The idea of modifying the TCP congestion control to improve throughput efficiency is not new and dates back at least to the work of Brakmo *et al* on TCP Vegas [6]. More recent work includes FAST TCP[7]. In both of these approaches the TCP transmission rate is adapted in response to changes in round-trip delay with the aim of maintaining queue occupancy at a small, but non-zero, value thereby improving link utilisation while also avoiding the packet drops associated with the probing action of AIMD congestion control. However, this involves a paradigm change in congestion control with a shift from use of packet drops as an indicator of congestion to use of delay as a congestion indicator. In the present paper, our aim is instead to remain within the well tested AIMD paradigm.

The idea of adapting AIMD parameters to reflect prevailing network conditions or achieve certain goals is also not new and a wide body of work exists on this topic [**?**], [**?**], [**?**],

[**?**], [**?**] The body of work most related to our proposal was developed in the context of wireless networks and error-prone links by Gerla and his co-authors [8]. They consider a TCP variant denoted TCP-Westwood that proposes modifying the AIMD backoff factor using an on-line estimate of the bandwidth available on a path. However, the strategy presented in the present paper differs from TCP-Westwood not only in the manner in which the AIMD backoff factor is adjusted (e.g. we make no attempt to estimate the per-flow packet rate of the bottleneck link and our adaptation scheme does not require complex adaptive filtering strategies) but also in our adjustment of the AIMD increase parameter according to $\alpha_i = 2(1 - \beta_i)$ in order to maintain network fairness and friendliness.

IX. CONCLUDING REMARKS

In this paper we present a new approach to the buffer-provisioning problem that retains the benefits of statistical multiplexing when many flows share a link while also achieving good link utilisation with small numbers of flows. We exploit the fact that the manner in which network buffers are provisioned is intimately related to the manner in which TCP operates. However, rather than designing buffers to accommodate the TCP AIMD algorithm, we suggest simple modifications to the AIMD algorithm to accommodate buffers in the network. We demonstrate that with only minor modifications to the AIMD algorithm, networks with small buffers can be designed that transport TCP traffic in a very efficient manner.

We argue that the benefits of modifying TCP according to the proposed adaptive AIMD algorithm are compelling: namely, a nearly complete decoupling of network provisioning from the details of the TCP AIMD congestion control algorithm. This is achieved without any negative impact on link utilisation or network fairness properties and results in lower network loss rates, and reduced sensitivity to reverse path queuing.

X. ACKNOWLEDGEMENTS

REFERENCES

[1] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. ACM SIGCOMM 2004*, 2004.
[2] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modelling TCP throughput: A simple model and its empirical validation," in *Proc. SIGCOMM 1998*, 1998.
[3] V. Jacobson, R. Braden, and L. Zhang, "TCP extensions for high-speed paths," in *IETF RFC 1185*, 1990.
[4] V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOMM 1988*, 1988.
[5] C. Villamizar and C. Song, "High performance TCP in ANSNET," *ACM Computer Communications Review*, vol. 24, no. 5, pp. 45–60, 1994.
[6] L. Brakmo, S. O'Malley, and L. Peterson, "New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM 1994*, pp. 24–35, 1994.
[7] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance." Caltech CS Report CaltechCSTR:2003:010, 2003.

[8] C.Casetti, M.Gerla, S.Mascolo, M.Sanadidi, and R.Wang, "TCP west-wood: Bandwidth estimation for enhanced transport over wireless links." Proc. ACM Mobicom, 2001.