H-TCP: A framework for congestion control in high-speed and long-distance networks

D.J. Leith, R.N. Shorten, Y.Lee
Hamilton Institute, NUI Maynooth

Abstract— **In this paper we present a new AIMD congestion control algorithm, H-TCP, that is suitable for deployment in high speed and long distance networks as well as conventional networks. H-TCP generalises the AIMD strategy to significantly improve the manner in which responsiveness and efficiency scale with path bandwidth-delay product and level of queue provisioning within the network. The performance of H-TCP is demonstrated via experimental measurements across a range of network conditions.**

I. Introduction

The topic of congestion control for high speed and long distance networks is currently the subject of much debate in the networking community. This debate has been driven by the realisation that TCP in its current form performs poorly in such networks, and has led to the development of several new congestion control algorithms. Examples include High-Speed TCP[1], Scalable TCP[2], FAST-TCP[3] and BIC-TCP[4]. Unfortunately, despite much effort, none of the proposed algorithms have yet provided compelling evidence to suggest that they are suitable for general deployment. The basic design requirement is that a number of important performance criteria must be satisfied by any new congestion control algorithm. To date, designing an algorithm that satisfies all of these criteria has been difficult to achieve.

While a consensus on what exactly constitutes a good set of performance criteria to support the design of new congestion control algorithms has yet to be agreed by the networking community, any new algorithm should have certain basic properties. For example, high-speed algorithms should be backward compatible with legacy TCP, their fairness properties should be well understood and they should strive to acquire and release bandwidth rapidly as network conditions change. The primary aim of the present paper is to relate each of these properties to the dynamics of the AIMD algorithm and use this knowledge to support a principled design of new congestion control algorithms.

II. Motivation: Brief Case Studies

It has been well documented in several case studies that current TCP congestion control algorithms can perform poorly over high-speed and long distance paths [2], [1], [4]. On such paths the bandwidth-delay product (BDP) can be very large. As a result flows may take an unacceptably long time to recover their window size after a congestion event. An obvious solution that addresses this problem is to redesign the congestion avoidance mode of TCP with the specific objective of keeping the time between congestion events small. This is the basic strategy that has been pursued by almost all researchers working in the area and is easily achieved by adjusting the additive increase parameter α to be greater for flows that travel over paths with large bandwidth-delay product. This solution, in combination with the design of additional modes of operation to ensure backward compatibility with legacy TCP, provides a mechanism for solving several of the issues associated with TCP for high speed networks.

The logic that orchestrates switching between the normal and high speed modes of operation can clearly be designed several ways. One approach is to use the current value of the congestion window $cwnd$ as an indicator of the path BDP. That is, the AIMD increase parameter α is increased as $cwnd$ increases thereby resulting in an additive increase algorithm that directly scales with the bandwidth-delay product. This is precisely the approach adopted in the High-Speed TCP [1] and Scalable TCP [2] proposals for high BDP networks. In addition to adjusting the AIMD increase parameter α as a function of $cwnd$, these proposals also increase the multiplicative decrease factor β to further increase the aggressiveness of a flow[1].

While such modifications might appear straightforward, and a logical amendment to TCP for high BDP paths, they dramatically alter the behaviour of networks of TCP flows.

Specifically, adjusting the AIMD parameters of a particular flow based on the flow state $cwnd$ creates a fundamental asymmetry in the network that can lead to undesirable network behaviour. Figure 1 illustrates the behaviour of High-Speed TCP following startup of a new flow. It can be seen that the convergence of the $cwnd$'s following this change in network conditions is extremely slow. The slow convergence arises from the aforementioned asymmetry; specifically, in the manner in which competing flows acquire and release

[1]On multiplicative decrease, $cwnd$ is updated to $\beta \times cwnd$. We use this definition of the backoff factor β throughout this paper.

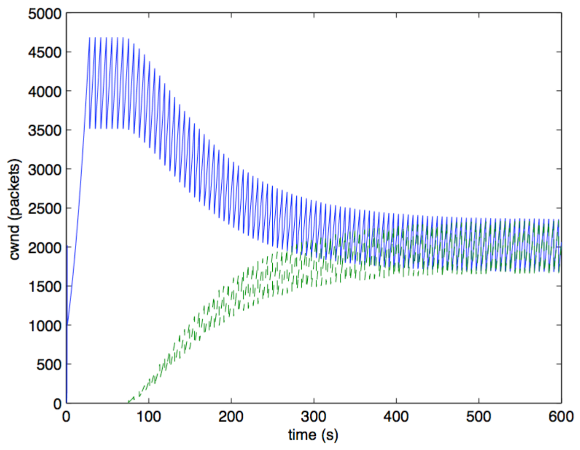


Fig. 1: Example of two High-Speed TCP flows illustrating slow convergence to fairness. (NS simulation, 500Mb bottleneck link, 100ms delay, queue 500 packets)

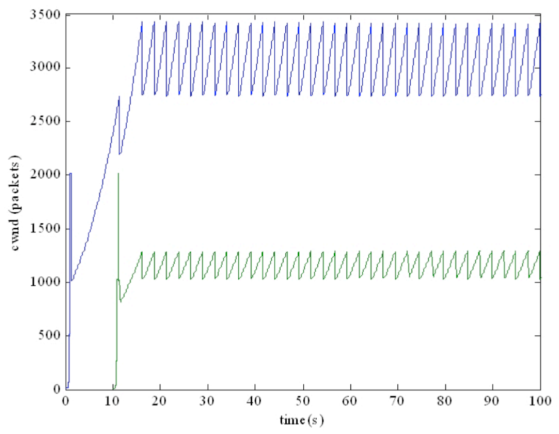


Fig. 2: Example of two Scalable TCP flows illustrating lack of convergence to fairness. (NS simulation, 500Mb bottleneck link, 100ms delay, queue 500 packets)

bandwidth. Paradoxically, flows with a large $cwnd$ release bandwidth (due to increased β) more slowly than those with a smaller $cwnd$, and acquire bandwidth at a faster rate (due to a larger α). Newly started flows are consequently placed at a disadvantage against established flows. [2]

In Scalable TCP, the AIMD increase parameter is selected to be proportional to the instantaneous $cwnd$ value. This has the effect of making the duration of a congestion epoch invariant with congestion window size. The impact, however, of this change on the behaviour of Scalable TCP flows following startup of a second flow is illustrated in Figure 2. It can be seen that the flows do not convergence to fairness. In fact, the Scalable TCP algorithm is a multiplicative-increase multiplicative-decrease (MIMD) algorithm as opposed to an additive-increase multiplicative-decrease (AIMD) one. It has been known for many years [5] that MIMD algorithms need not converge to fairness under drop-tail queueing.

Evidently, TCP congestion control algorithms must be designed with a number of multiple objectives in mind. In particular, adequately addressing all of the following issues seems to be a minimum requirement in the design of any TCP protocol that is suitable for deployment in high BDP networks.

(i) **Friendliness with legacy TCP.** In low speed environments new variants of TCP should behave similarly to legacy TCP traffic. In high-speed environments, legacy TCP traffic should not be completely starved of bandwidth.

(ii) **Efficiency.** TCP should ensure efficient use of network resources. In particular, TCP flows should strive to efficiently utilise available bandwidth.

(iii) **Fairness.** We refer here to fairness between competing flows that utilise the *same* TCP algorithm. We expect that new variants of TCP should be not be significantly more unfair than current TCP (e.g. RTT unfairness should be no worse than that exhibited by current TCP algorithms).

(iv) **Responsiveness.** That is, how quickly flows respond to changes in network conditions such as the starting or stopping of a network flow.

Our objective in the remainder of this paper is to devise a congestion control algorithm for high-speed networks that performs well according to all of the above criteria.

III. Network Dynamics

The AIMD algorithm that is used in TCP has two key features that underpin its convergence behaviour. Firstly, flows increase their $cwnd$ at the same rate [3] and seize bandwidth at the same rate. That is, we have symmetry in the flow increase rates. Secondly, the backoff mechanism is multiplicative. Hence, following congestion, flows with a larger $cwnd$ will reduce this $cwnd$ by more, in absolute terms, than flows with a smaller $cwnd$. That is, larger flows yield more bandwidth than smaller flows. Flows with smaller $cwnd$ thereby gain a certain advantage over flows with larger $cwnd$, and it is this that enables flows with small $cwnd$ to seize bandwidth from flows with large $cwnd$ until balance is reached in the network.

We can express this mathematically as follows. Let $w_i(k)$ denote the $cwnd$ value of flow i immediately before backoff

[2]In the example shown the second flow experiences a packet drop early in slow-start. While slow-start might be amended to improve start-up performance, as noted previously this does not remove the difficulties associated with the slow responsiveness of the congestion avoidance algorithm.

[3]We assume for the moment that flows have the same round-trip time. This simplifies the discussion, but the arguments presented carry over to the more general case, see later.

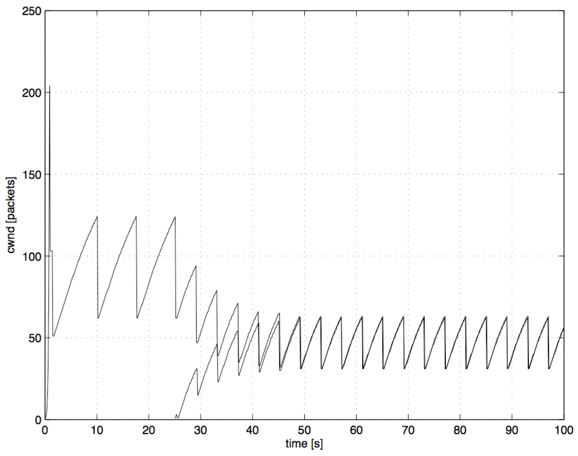


Fig. 3: Illustration of TCP convergence rate (NS simulation, $\alpha_i = 1$, $\beta_i = 0.5$, dumb-bell with 10Mbs bottleneck bandwidth, 100ms propagation delay, 40 packet queue).

with k indexing the congestion events. Let $\alpha_i(k)$ and β be the AIMD increase and decrease parameters of the i'th flow, and let $T(k)$ be the interval between congestion events k and $k+1$. We measure $T(k)$ in seconds and define $\alpha_i(k) = (w_i(k+1) - \beta w_i(k))/T(k)$ which is simply the effective increase rate in packets/s. Note one can often approximate $\alpha_i(k)$ by $1/RTT_i$ where RTT_i is the round-trip time of flow i. From the AIMD algorithm we have that

$$w_i(k+1) = \beta w_i(k) + \alpha_i(k)T(k) \tag{1}$$

And for two flows i and j that experience congestion it follows that

$$w_i(k+1) - w_j(k+1) = \beta(w_i(k) - w_j(k)) \tag{2}$$

After n congestion events we have

$$w_i(k+n) - w_j(k+n) = \beta^n(w_i(k) - w_j(k)) \tag{3}$$

Since $\beta < 1$, we can see that the congestion windows w_i and w_j must eventually converge to the same values.

It follows immediately from this analysis that the number of congestion epochs that flows take to converge is determined by the backoff parameter β. With $\beta = 0.5$, after 2.3 congestion epochs the difference in flow congestion windows will have reduced to less than 20%, and after approximately 4 congestion epochs to less than 5%. This behaviour is illustrated, for example, Figure 3.

While β determines the number of congestion epochs for convergence, the duration of the congestion epochs obviously depends on both β and α. Increasing β and/or increasing α reduces the congestion epoch duration. Note, however, that adjusting β to reduce congestion epoch duration results in a corresponding (exponential) increase in the number of congestion epochs before convergence.

A second key insight that can be gained from the foregoing analysis is that symmetry in the increase rates of the competing flows plays a central role in ensuring convergence to a fair equilibrium. In one sense this observation is unsurprising - it has been known for many years that asymmetry created by differences in RTT (leading directly to differences in increase rate) creates unfairness. However, other types of asymmetry, such as that created by the High-Speed TCP algorithm where the increase rate is coupled to $cwnd$ size, can evidently induce both unfairness and complex transient behaviour.

Comment: Different RTTs. The foregoing analysis can be extended to include flows with different round-trip times as follows. Let $z_i = w_i/\alpha_i$ be the congestion window normalised by the increase rate. It follows that $z_i(k+1) = \beta z_i(k) + T(k)$ and $z_j(k+1) = \beta z_j(k) + T(k)$. Hence, $z_i(k+1) - z_j(k+1) = \beta(z_i(k) - z_j(k))$ similarly to (2). In equilibrium $z_i = z_j$; that is, $w_i/w_j = \alpha_i/\alpha_j$. Recalling that $\alpha_i \approx 1/RTT_i$ we have $w_i/w_j \approx RTT_j/RTT_i$.

Comment: Network equilibrium. More generally, in equilibrium we have that $w_i = \frac{\alpha_i T(k)}{1-\beta_i}$. Hence, for two flows i and j, $\frac{w_i}{w_j} = \frac{\alpha_i(1-\beta_j)}{\alpha_j(1-\beta_i)}$. This analysis indicates that fairness is ensured when the ratio $\frac{\alpha_i}{1-\beta_i}$ is the same for all flows. Since in the case of standard TCP we have that $\alpha_i = 1$, $\beta_i = 0.5$, the requirement for fairness with standard TCP flows is that $\alpha_i = 2(1-\beta_i)$.

Comment: Symmetry. It is easily observed that the foregoing analysis is not only valid for conventional AIMD networks, but also extends to any network type in which flows update their congestion windows according to

$$w_i(k+1) = \beta w_i(k) + k_i f(T(k)), \tag{4}$$

where k_i is some constant, and $f(T(k))$ is any function that is the same for all flows i.e ensuring symmetry between flows.

Comment: Unsynchronised drops. In equation (3) we have implicitly assumed that flows i and j both experience drops at every congestion event i.e that they flows are synchronised. The analysis can be extended to unsynchronised flows but at the price of a more involved development, see [6], [7] for details. The qualitative insights relating to the influence of the AIMD parameters remain unchanged in the unsynchronised case.

IV. Generalising AIMD

We consider generalising the AIMD algorithm by allowing the increase parameter to vary as a function of the elapsed time since the last congestion event. Specifically, if we let Δ denote the time in seconds that has elapsed since the last congestion event experienced by a flow, we adjust the AIMD increase parameter according to some function which we denote $\bar{\alpha}(\Delta)$. To provide backward compatibility with legacy

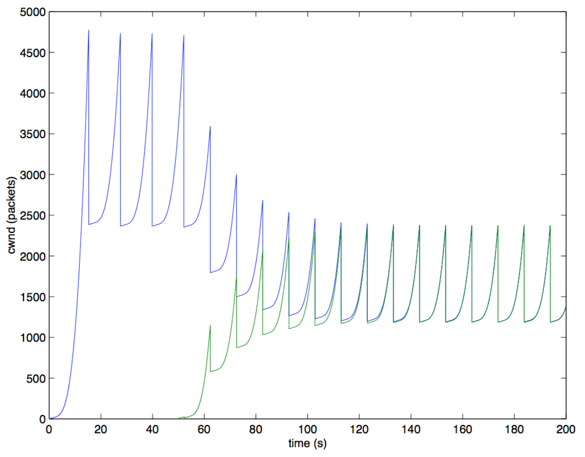


Fig. 4: Convergence of flows where AIMD increase parameter is varied as a function of elapsed time since last backoff according to (7). (NS simulation, 500Mb bottleneck link, 100ms delay, queue 500 packets)

TCP flows we consider adjusting the increase parameter as follows

$$\bar{\alpha}(\Delta) \quad = \quad \begin{cases} 1 & \Delta \leq \Delta^L \\ \bar{\alpha}^H(\Delta) & \Delta > \Delta^L \end{cases} \qquad (5)$$

where Δ^L is the threshold for switching from standard/legacy operation to the new increase function, $\bar{\alpha}^H(\Delta)$. Here we use the overstrike on $\bar{\alpha}$ to indicate that this is the increase rate measured in packets/RTT as opposed to packets/sec; that is, we would update $cwnd$ to $cwnd + \bar{\alpha}(\Delta)/cwnd$ on receipt of each TCP ACK packet. The choice of function $\bar{\alpha}^H(\Delta)$ is governed by the rate at which bandwidth should be acquired. Note that when $\bar{\alpha}^H(\Delta) = 1$ we recover the standard AIMD algorithm.

We can immediately make the observation that, because the adjustment is based on time since the last back-off, flows already in high speed mode are not awarded a long-term advantage over newer flows. When drops are synchronised, flows grab bandwidth at the same rate. When drops are not synchronised, flows grab bandwidth at the same rate on *average*, provided their probability of backing off on congestion is the same.

It follows from the results of the previous section that the number of congestion epochs for convergence is determined by the backoff factor β. This is illustrated in Figure IV which shows convergence following startup of a second flow. The backoff factor here is the standard TCP value of 0.5 and it can be seen that convergence takes approximately 4 congestion epochs, as with standard TCP (Figure 3).

Note that from a user perspective, the issue of interest is the convergence rate in seconds rather than the convergence rate measured in congestion epochs. To improve the responsiveness of a network we therefore need to choose the increase function $\bar{\alpha}^H(\Delta)$ such that the duration of the congestion epochs remains reasonably small as the bandwidth-delay product on a path increases. The congestion epoch duration is given by the solution to the implicit equation

$$\alpha(k)T(k) \quad = \quad w(k+1) - \beta w(k) \qquad (6)$$

where $\alpha(k) = \frac{\int_0^{T(k)} \bar{\alpha}(t)dt}{T(k)RTT}$. Choosing, for example,

$$\bar{\alpha}^H(\Delta) \quad = \quad 1 + 10(\Delta - \Delta^L) + (\frac{\Delta - \Delta^L}{2})^2. \qquad (7)$$

and $\Delta_L = 1s$ yields a polynomial increase in $cwnd$ over a congestion epoch and by adjusting the polynomial we can adjust the congestion epoch duration. In particular, the increase function (7) yields the congestion epoch duration for a single flow, as a function of congestion window size, shown in Figure 5. It can seen that the congestion epoch duration is around 10s for a bandwidth-delay product of 10,000 packets, yielding a convergence time to 20% of final value of 25s, and a convergence time to 5% of final value of 40s. This choice of convergence time is somewhat arbitrary and other choices of increase function would yield different values. However, the precise responsiveness requirement in future networks is currently not well defined and so we leave this, and the associated specific choice of increase function, as a question for further debate. The increase function (7) is used in the rest of this paper merely as an illustrative example that yields convergence times that seem reasonable.

Detailed experimental results measuring the performance of the proposed increase law are presented in later sections. Here, however, we can briefly make some preliminary observations. Figure IV illustrates the congestion window evolution obtained using the increase function (7).

(i) **Response function.** It is readily shown that (7) has a response function similar to that proposed by Floyd [1].

(ii) **Friendliness.** Define the effective linear increase as $(w(k+1) - w(k))/T(k)$. This is the value of increase parameter in conventional AIMD that corresponds to the increase function (7). The effective increase parameter in units of packets/RTT is shown in Table I for a range of bandwidth-delay products. This an indication of the number of standard TCP flows (neglecting statistical multiplexing of backoffs) whose aggregate would be equivalent to a flow using increase function (7).

(iii) **RTT unfairness.** Using a similar argument to that in the previous section, namely working in terms of normalised congestion windows, we can see that the degree of RTT unfairness is similar to that with standard TCP.

(iv) **RTT scaling.** Recall that for standard TCP we have that the effective increase rate is inversely proportional to round-trip time. A similar situation holds for the increase function (7). In both cases, we note that the additive increase law can be effectively made invariant with round-trip time by simply scaling α by the round-trip time[4] in which case the convergence time also becomes invariant with round-trip time. Such RTT scaling also can be expected to reduce the level of unfairness experienced between competing flows with different round-trip times.

Congestion window (packets)	Effective number of standard TCP flows		
	10ms RTT	100ms RTT	250ms RTT
10	1	1	1
100	1	2	5
1000	3	12	22
2000	4	19	32
5000	8	33	55
10000	12	49	82
20000	19	72	123
50000	32	122	208

TABLE I: Effective increase parameter (packets/RTT) vs window size.

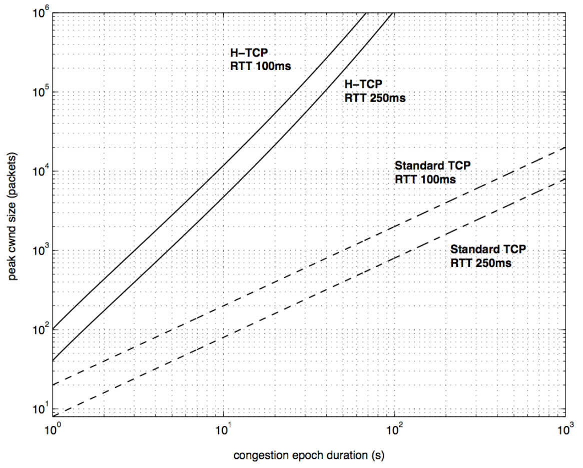


Fig. 5: Peak window size achieved vs duration of congestion epoch with standard TCP and with α^H selected according to (7)

V. Adaptive Backoff

While the AIMD increase parameter α has a relatively straightforward impact on network behaviour (provided care is taken to maintain the appropriate symmetry) the influence of the backoff factor β is more complicated. With regard to responsiveness, changes to β impact both the congestion epoch duration and the number of congestion epochs before convergence. Further, the choice of backoff factor has a significant impact on throughput efficiency and queue requirements within the network.

A. Throughput Efficiency

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

$$R(k)^- \quad = \quad \sum_i^n \frac{w_i(k)}{T_i + \frac{q_{max}}{B}} = B \tag{8}$$

where 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. 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

$$R(k)^+ \quad = \quad \sum_i^n \frac{\beta_i(k) w_i(k)}{T_i} \tag{9}$$

under the assumption that the bottleneck buffer empties[5]. 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,

$$\beta_i \geq \frac{T_i}{T_i + \frac{q_{max}}{B}} = \frac{RTT_{min,i}}{RTT_{max,i}}. \tag{10}$$

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 networks employing standard TCP, namely $\beta_i = 0.5$, it follows $q_{max} \geq BT_i$. This is, at least in part, the origin of the bandwidth-delay product (BDP) rule. The BDP rule is, however, only one of many strategies that could be adopted to ensure that Equation (10) is satisfied. In particular, for any given queue size q_{max} one may simply set

$$\beta_i = \frac{RTT_{min,i}}{RTT_{max,i}} \tag{11}$$

for all i; thereby ensuring that $R(k)^+ = R(k)^-$ for all k. The effect of this AIMD modification can be seen in Figure 6. In this example the queue provisioning is less than the bandwidth-delay 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

[4]It is of course prudent to restrict such scaling to lie in some interval, say [0.5,10], to prevent misbehaviour on paths with very short or very long round-trip times.

[5]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%.

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, rather than designing buffers to accommodate the TCP AIMD algorithm, we modify the AIMD algorithm to accommodate the buffers in the network. This is particularly relevant in high-speed environments where queues are often relatively small relative to the delay-bandwidth product.

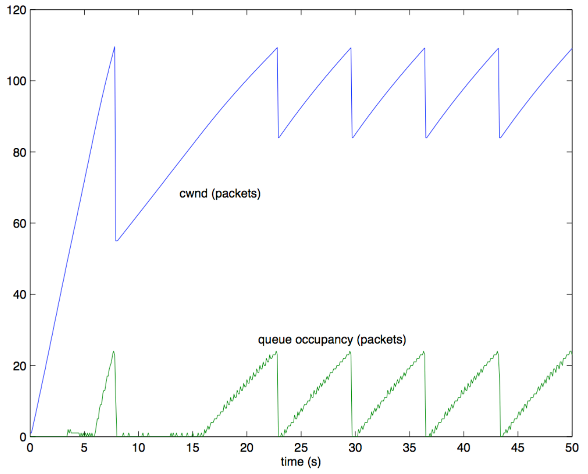


Fig. 6: Congestion window and queue occupancy time histories with adaptive backoff. The delay bandwidth product is 85 packets. (NS simulation, bandwidth 10Mbs, RTT 100ms, queue 25 packets)

.

Comment: Primal-Dual Strategies. The BDP-rule and the adaptive backoff strategy proposed above may be thought of as, respectively, primal and dual methods. The BDP strategy requires correct sizing of the network buffers. The adaptive backoff strategy requires that each source adjust its backoff factor appropriately.

Comment: Statistical Multiplexing. The BDP rule is derived from consideration of link utilisation in the worst case where all flows backoff at each congestion event. This situation applies when only a single TCP flow is present and 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%). Studies that we have carried out indicate that the statistical multiplexing gain with adaptive backoff behaves similarly to that with standard TCP; that is, the buffering requirement is approximately inversely proportional to $\sqrt{n}$ where n is the number of competing flows (see Appenzeller *et al*[8]).

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. We have found that minimum and maximum RTT are, however, relatively straightforward to estimate.

B. *Efficiency vs Responsiveness*

It is important to be aware that adjusting the flow backoff factors can be expected to have an impact on network responsiveness. Indeed, within the AIMD paradigm the foregoing analysis seems to indicate that a tradeoff of sorts exists between responsiveness and link utilisation. Increasing the backoff factor β to improve efficiency decreases responsiveness (as measured by number of congestion epochs before convergence[6]). Conversely, decreasing the backoff factor increases responsiveness but decreases efficiency, at least when queues are small. Note that large queues resolve this tradeoff 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 bandwidth-delay product[7] 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.

C. *Maintaining Fairness*

Adjustment of the flow backoff factors β_i can lead to unfairness between competing flows. The analysis in Section IV shows, however, that fairness can be restored by adjusting α_i such that $\alpha_i = 2(1 - \beta_i)$[8]. This yields the following

[6]Increasing β decreases the congestion epoch duration while increasing the number of congestion epochs for convergence. Typically, however, it is the latter effect that dominates as the number of epochs increases exponentially with β whereas the epoch duration only decreases linearly.

[7]This is the worst case buffering requirement and is reduced by statistical multiplexing of flow backoffs.

[8]This yields fairness in the sense that flow congestion windows are equal. We use this notion of fairness as it naturally arises in describing network behaviour in terms of congestion events. Other forms of fairness could, of course, also be used and the backoff scheme modified appropriately.

adaptive backoff algorithm

$$\begin{aligned}\beta_i(k) &= \frac{RTT_{min,i}(k)}{RTT_{max,i}(k)};\\ \alpha_i(k) &= 2(1-\beta_i(k))\end{aligned}$$

The effect of adjusting α_i in this manner is illustrated in Figure 7, 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.

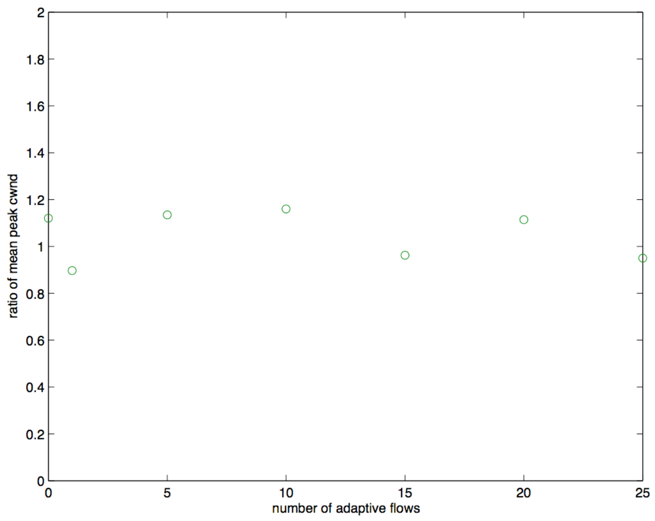


Fig. 7: Fairness of adaptive and conventional TCP flows vs number of adaptive AIMD flows (out of 25 flows in total, remaining flows are conventional TCP flows). NS simulation, 155Mb bottleneck link, 120ms delay, queue 125 packets. Network includes background web traffic of approximately 1% link bandwidth.

D. Multiple Bottlenecks

It is easy to see that the proposed adaptive backoff strategy readily extends to paths with multiple congested links. Assume, for the moment, the worst case from a link utilisation viewpoint) situation where flows are synchronised. At congestion we have that

$$R(k)^- = \sum_{i=1}^{n} \frac{w_i(k)}{\sum_{j=1}^{m}(T_i + q_j(k)/B_j)} = B \quad (12)$$

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

$$\beta_i(k) = \frac{T_i}{T_i + \sum_{j=1}^{m} \frac{q_j(k)}{B_j}} = \frac{RTT_{min,i}}{RTT_{max,i}} \quad (13)$$

then after backoff,

$$R(k)^+ = \sum_{i}^{n} \frac{T_i}{T_i + \sum_{j=1}^{m} \frac{q_j(k)}{B_j}} \frac{w_i(k)}{T_i} = B \quad (14)$$

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: 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. To implement the backoff calculation in (13), we use $\frac{RTT_{min,i}}{RTT_{max,i}}$ 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

$$RTT_{max,i} = RTT_{min,i} + a(RTT_{max,i} - RTT_{min,i}) \quad (15)$$

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

VI. H-TCP

We can directly combine the foregoing techniques for adjusting the AIMD increase rate and backoff factor to improve the responsiveness and efficiency of TCP in high-speed networks. This yields the following complete algorithm.

(a) On each acknowledgement:

$$\begin{aligned}\bar{\alpha} &\leftarrow 2(1-\beta)\bar{\alpha}(\Delta) \quad (16)\\ cwnd &\leftarrow cwnd + \bar{\alpha}/cwnd \quad (17)\end{aligned}$$

(b) On each congestion event:

$$\begin{aligned}\beta &\leftarrow \frac{RTT_{min}}{RTT_{max}}, \quad \beta \in [0.5, 0.8], \quad (18)\\ cwnd &\leftarrow \beta \times cwnd \quad (19)\end{aligned}$$

where

$$\bar{\alpha} = \begin{cases} 1 & \Delta \le \Delta^L \\ 1 + 10(\Delta - \Delta^L) + (\frac{\Delta - \Delta^L}{2})^2 & \Delta > \Delta^L \end{cases} \quad (20)$$

and Δ is the time in seconds since the last congestion event, $\Delta_L = 1s$ and $\frac{RTT_{min}}{RTT_{max}}$ is the ratio of minimum and maximum RTT's of the flow.

As discussed previously, by scaling $\bar{\alpha}$ with round-trip time we can optionally make the increase rate effectively invariant with round-trip time, in which case convergence time is also invariant with round-trip time. RTT unfairness between competing flows is also mitigated when RTT scaling is employed.

VII. Experimental Evaluation

It is well known that even small modifications to the basic TCP algorithm can have a large impact on network behaviour. Although insight from analysis can assist in developing new algorithms and avoiding known pitfalls, it is essential to carry out experimental tests to help us gain confidence in the correct operation of an algorithm. To this end, we have defined a collection of benchmark tests that exercise many of the major aspects of TCP congestion control behaviour.

As we have already discussed, the basic performance of any proposed congestion control algorithm, can be captured to some degree by evaluating its performance according to the following criteria; fairness (between like flows), friendliness (fairness with legacy TCP flows), efficiency (use of available network bandwidth) and flow responsiveness. Rather than trying to define a single metric to capture performance, our approach is to devise a number of tests to measure protocol behaviour according to the above criteria. In this context we use the behaviour of standard TCP as the baseline against which to assess the performance of any new protocol. Since the behaviour of TCP is known to be dependent on factors including queue provisioning, link bandwidth, round-trip time and difference in round-trip time (for competing flows) we evaluate the impact of these factors on performance. Number of flows is also known to be an important factor in TCP performance. Owing to space constraints, in this paper we only present results for two competing flows; our experience indicates that this is sufficient to highlight many issues of interest (RTT unfairness, convergence times, TCP friendliness etc). We have obtained similar results for > 2 flows.

Although it is not our objective to evaluate and comment on other recent alternative high BDP protocol proposals, results are included for High-Speed TCP, Scalable TCP, BIC-TCP and FAST TCP to properly benchmark H-TCP.

A. *Experimental Setup*

Our test network consists 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. The servers are specified to ensure that throughput is limited by network bandwidth rather than NIC performance. We use a Linux 2.6.6 kernel with web100 extensions. This baseline kernel includes implementations of High-Speed TCP and Scalable TCP. The kernel has been modified by us to implement the H-TCP algorithm and to import a beta release implementation of FAST TCP provided by Caltech and a release of Bic-TCP provided by the group at North Carolina[9]. Hence, all comparisons are carried out using a common network stack implementation to provide a level playing field for comparing congestion control algorithms.

It is known [10] that the efficiency of the existing network stack implementation, in particular the SACK processing overhead, becomes the main constraint on performance on large bandwidth-delay product paths. The network stack has therefore been modified to use a more efficient SACK processing algorithm with computation burden that scales with number of lost packets rather than with number of packets in flight. A number of features of the Linux implementation that are known to degrade performance in high-speed environments are also disabled (packets-in-flight limiting, partial rather than full undo). We use delayed acking and implement appropriate byte counting as per RFC 3465. For further details, see patches available at www.hamilton.ie/net/.

The reported measurements are averages taken over at least 5 test runs each of at least 10 minutes duration.

B. *Flows with Same RTT*

We consider initially the case where all TCP flows operate the same congestion control algorithm, have the same round-trip time and share a common bottleneck link. Although this is a simple test case, it provides a useful starting point for evaluating algorithm performance.

Under these conditions, we expect that competing flows will, on average, attain the same average throughput. Figure 8 plots measurements of the ratio of flow goodputs as the network propagation delay is varied from 16ms to 162ms. Results are shown for a 10Mbs bottleneck link and for a 250Mbs link, roughly corresponding to low-speed and high-speed network environments (similar results are obtained for other link speeds). In these plots the network buffers are sized at 20% of the BDP. We have also studied other buffer sizes, but space constraints limit what we can include and the 20% BDP results seem particularly relevant as high-speed paths commonly have relatively small queues. Under these conditions, the standard TCP congestion control algorithm consistently ensures that each flow achieves the same (to within less than 5%) average throughput. However, the measurements indicate that many of the proposed protocols exhibit substantial unfairness under the same conditions. While both FAST-TCP and Scalable-TCP display very large variations in fairness, BIC-TCP and HS-TCP also display significant levels of unfairness. Our purpose in this paper is not to analyse or explain the behaviour of these

[9]We note that the implementation of Bic-TCP included in the standard Linux 2.6.6 kernel distribution is known [9] to be incorrect (this has subsequently been corrected). In our tests we use a corrected implementation based upon the original Linux patch developed by the BIC-TCP authors.

algorithms. We note briefly, however, that Scalable TCP is a multiplicative-increase multiplicative-decrease algorithm and as such it is known that it may not to converge to fairness. The unfairness exhibited by HS-TCP and Bic-TCP is associated with slow convergence following startup of a second flow - the results presented are for 10 minute test runs and convergence times frequently exceed this duration. These HS-TCP results are consistent with the simulation results reported in [11] , while the slow convergence of Bic-TCP seems consistent with data on the Bic-TCP authors web site. Convergence times are discussed in more detail below.

The throughput efficiency of standard TCP depends upon the level of queue provisioning. When queues are sized at the bandwidth-delay product, queues do not empty following backoff of the AIMD algorithm and link utilisation is 100%. For smaller queue sizes the link utilisation decreases (as we are considering only two flows, statistical multiplexing effects are insignificant). Figure 9(a) plots measurements of aggregate goodput vs queue provisioning. The results shown are for two TCP flows in a network with 82ms propagation delay and 100Mbs bottleneck bandwidth (we obtain similar results for other network conditions). As a validation check, also plotted on Figure 9(a) is the efficiency for standard TCP predicted by NS simulations. It can be seen that the experimental and simulation throughputs are in good agreement. Note that data goodput is shown and hence the efficiency cannot exceed 94% owing to the ethernet framing and packet header overheads.

It can be seen that, as expected, for standard TCP the throughput efficiency falls as the queue size is reduced. While the theoretical analysis indicates a minimum throughput efficiency of 75% (or 70% when framing and header overheads are taken into account) when the queue size is zero, in practice it can be seen that the efficiency rapidly decreases as the queue size fall below around 3% of the BDP (or less than about 8% in the case of FAST TCP). Further investigation indicates that this is associated with microscale packet bursts (associated with ACK clocking and scheduling granularity within the end hosts) flooding the queue. All of the protocols proposed for high-speed networks achieve greater throughput efficiency than standard TCP, with H-TCP attaining the maximum achievable throughput for queue sizes above 25% BDP in line with the analysis in Section V-B.

Another issue related to throughput efficiency is the packet loss overhead generated by a congestion control algorithm. Since TCP is a reliable transport protocol, packet losses result in the retransmission of the packets concerned. Trivially, an algorithm can attain maximum throughput by sufficiently overloading the network but this is unacceptable if it entails unduly high packet loss rates. Figure 9(b) shows the measured packet retransmission overhead versus queue provisioning. With the notable exception of FAST-TCP, it can be seen that as expected the new protocols behave similarly and uniformly carry a greater overhead than standard TCP. The packet loss overhead of FAST-TCP is observed to be strongly dependent on queue provisioning: with small queues FAST-TCP has the largest overhead, while for large queues FAST-TCP has smallest overhead.

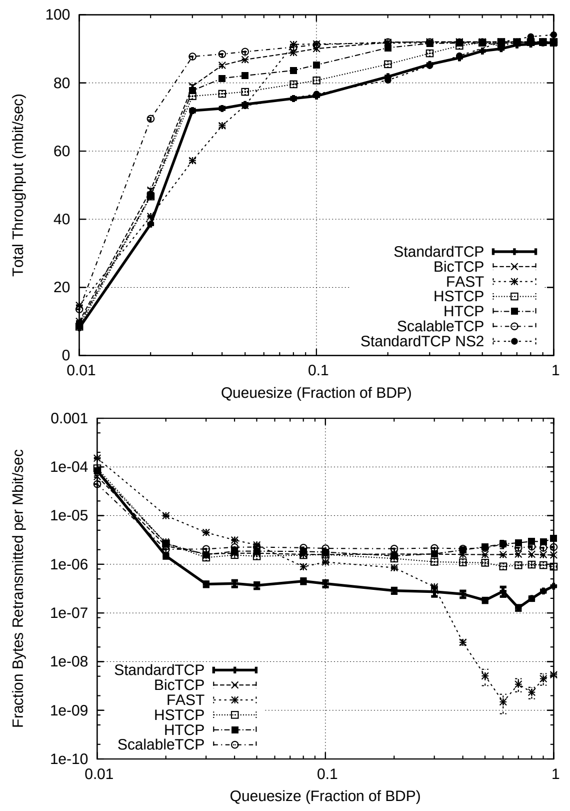


Fig. 9: Throughput and packet loss overhead vs queue provisioning (as a proportion of the bandwidth-delay product) for two TCP flows. Both flows have end-to-end round-trip propagation delays of 82ms and the bottleneck bandwidth is 100Mb/s; BDP is 683 packets.. Throughput measured is data payload and excludes ethernet framing and TCP/IP header overhead; maximum achievable data throughput is therefore 94% link bandwidth.

Figure 10 shows measurements of the convergence time following startup of a second flow. The convergence time is plotted versus path propagation delay (both flows have the same propagation delay in this experiment) and results are presented for link rates of 10Mb/s and 250Mb/s. It can be seen that HS-TCP, Bic-TCP ans Scalable TCP exhibit slow convergence, in line with the discussion in Section 2 above. The convergence time of H-TCP remains essentially constant at 25s for round-trip times above 40ms and this reflects the use of RTT scaling of the increase function. Convergence times are faster for smaller RTT's owing to the [0.5,10] clamp on RTT scaling used in these tests (these clamp values are used only to illustrate the impact of clamping).

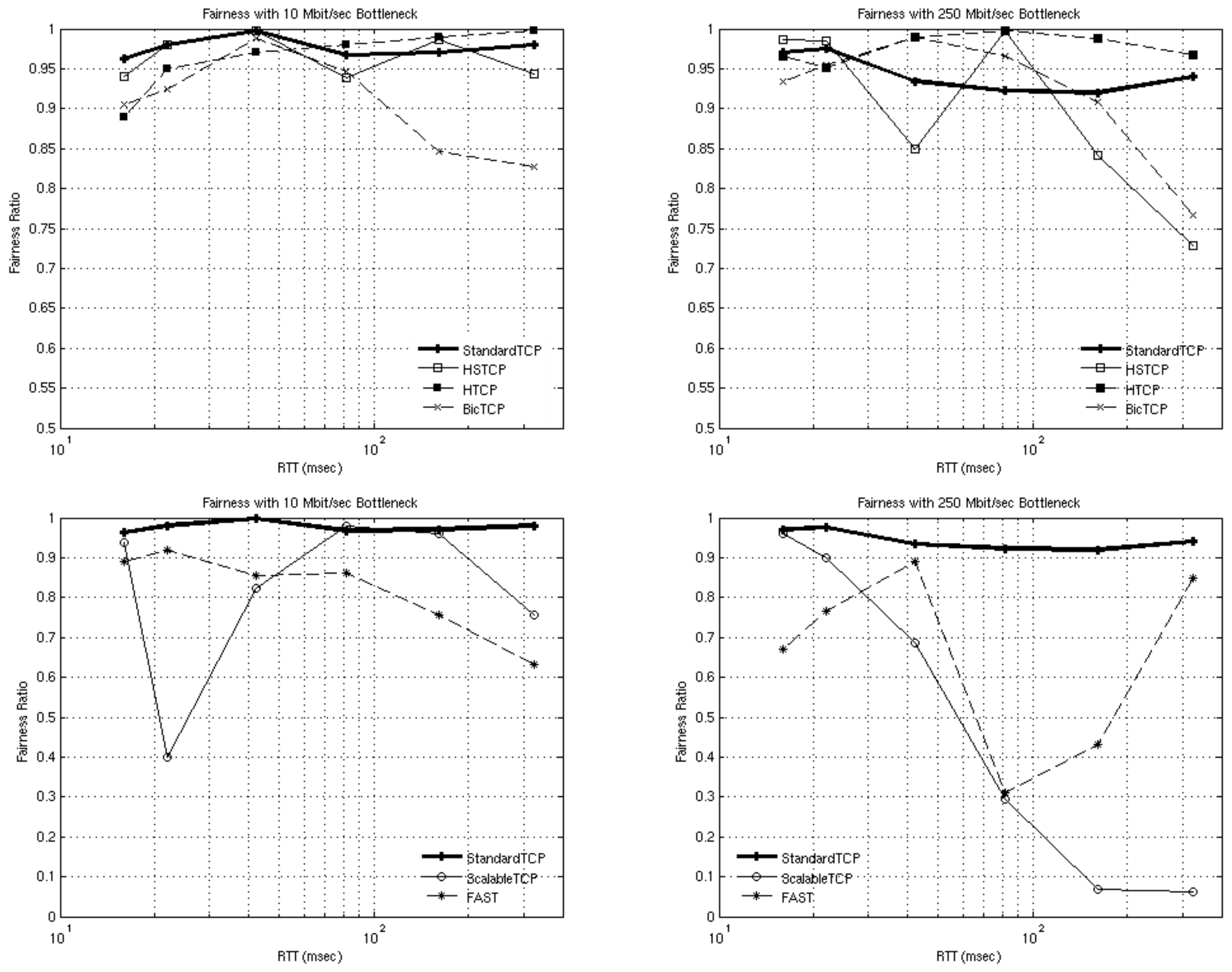


Fig. 8: Ratio of throughputs of two flows under symmetric conditions (same propagation delay, shared bottleneck link, same congestion control algorithm) as path propagation delay is varied. Results are shown for 10Mbit/sec and 250Mbit/sec bottleneck bandwidths. The bottleneck queue size is 20% BDP. Observe that while standard TCP and H-TCP are essentially fair (the competing flows achieve, to within 5%, the same average throughput) under these conditions.

C. Flows with Different RTT

The manner in which competing TCP flows interact with each other is known to depend on their round-trip times. In particular, flows with shorter round-trip times are able to gain greater throughput than those with longer round-trip times. In this section we present measurements obtained with competing flows operating the same congestion control algorithm but having different round-trip times.

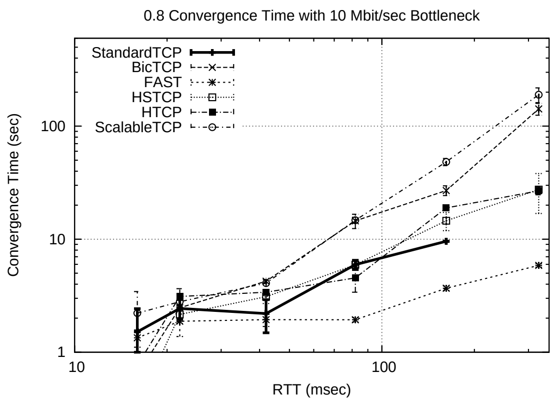
Figure 11 plots measurements of the ratio of the goodputs of two flows as the network propagation delay of the second flow is varied from 16ms to 162ms (the delay of the first flow is held constant at 162ms). Results are shown for a 10Mbs bottleneck link and for a 250Mbs link.

Comment: As a validation check, also plotted on Figure 11 are the throughputs for standard TCP predicted by the theoretical analysis in [7]. Namely, the throughput given by the expression

$$\bar{u_i} = \frac{\alpha}{\lambda_i T_i (1-\beta))} \tag{21}$$

where α and β are the AIMD increase and decrease parameters (having values of 1 and 0.5, respectively, for standard TCP), T_i is the round-trip time of flow i, λ_i is the synchronisation factor i.e. the proportion of network congestion events at which the flow i experiences a packet drop and backs off its $cwnd$ (λ_i is estimated from the measured $cwnd$ time histories). It can be seen that the experimental and theoretical throughputs are in good agreement.

With the exception of H-TCP it can be seen from Figure 11 that all of the new proposals exhibit significantly greater RTT unfairness than standard TCP. The degree of unfairness



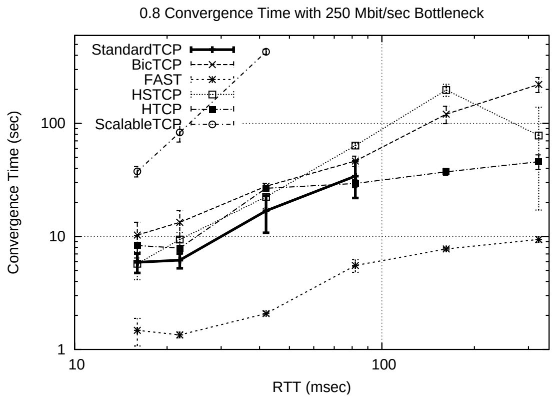
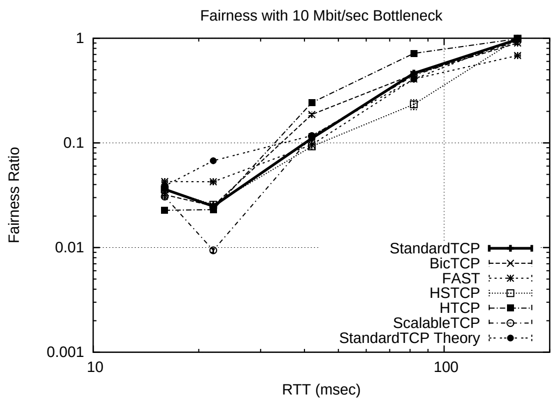


Fig. 10: 80% Mean 80% convergence time following startup of a second flow. Results are shown for 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidths. Both flows have same RTT. Queue size is 20% BDP. Missing points along the ordinate axis indicate that the flows did not converge to within the 80% fairness ratio over the 10 minute duration of the test – this is especially evident with Scalable-TCP and standard TCP for long delays at 250Mb/sec.



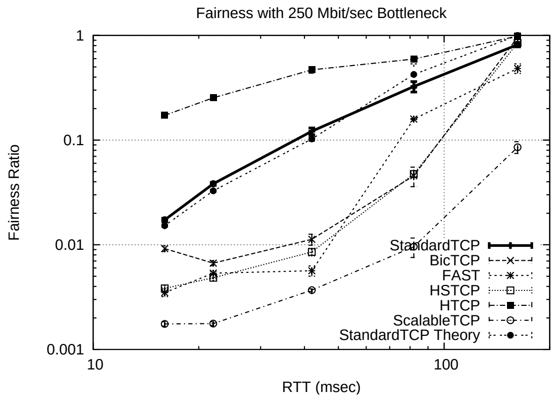


Fig. 11: Ratio of throughputs of two competing flows as the propagation delay of the second flow is varied. Results are shown for 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidths. Flow 1 has RTT of 162ms, the RTT of Flow 2 is marked on the x-axis of the plots. Queue size is 20% BDP.

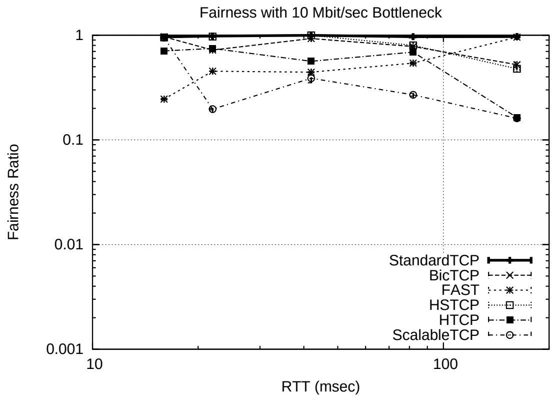
can be nearly an order of magnitude greater than that with standard TCP and is such that long round-trip time flows may be essentially starved of bandwidth. With regard to HS-TCP, these RTT unfairness measurements are in line with results reported elsewhere [4] and can be understood by noting that the unfairness in standard AIMD associated with flows having different RTTs is amplified in HS-TCP by the adjustment of the AIMD increase parameter as a function of $cwnd$. The RTT unfairness between competing H-TCP flows is somewhat less than that of standard TCP owing to the RTT scaling used (without RTT scaling the measured RTT unfairness is very similar to that of standard TCP).

D. TCP Friendliness

In high-speed network conditions where legacy TCP flows are unable to make efficient use of the available bandwidth, we assume that a degree of unfairness is permitted. However, it is important that modifications to the TCP congestion control algorithm do not lead to an excessive level of unfairness when operating in a heterogeneous environment that includes legacy TCP flows. Figure 12 plots measurements for two competing flows where one flow is standard TCP and the second uses a modified congestion control algorithm. It can be seen that Scalable TCP and FAST TCP exhibit the greatest degree of unfairness in both low and high-speed conditions.

E. Scope of our results

Our principal consideration in this paper is networks that employ drop-tail queueing as this is prevalent in current networks. The experimental results that we have presented are for a number of precisely defined network scenarios. Clearly, these results do not amount to proof of the correct operation of any algorithm studied. However, we argue that such tests are nevertheless useful as a screening tool that is both systematic and provides useful insight. The tests used are, for example, sufficient to reveal many of the issues already known to affect recent TCP proposals.



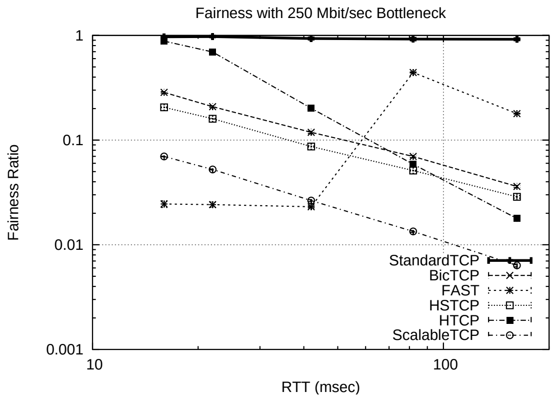


Fig. 12: Ratio of throughputs of competing New-TCP and standard TCP flows as path propagation delay is varied. Results are shown for 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidths. Both flows have the same RTT. Queue size is 20% BDP.

With regard to H-TCP, our experimental measurements do seem to be consistent with the analysis presented in earlier sections and therefore provides a degree of confidence in the viability of the proposed approach. We argue that the results presented justify further testing of the H-TCP framework, in particular wider testing in live network environments. We have already carried out initial tests over production paths between Dublin (Ireland), CERN (Geneva, Switzerland), Starlight (Chicago) and SLAC (California). However, these results are for a single flow and it is clear from our lab tests that it is important to perform tests between a number of competing flows as it is only under these conditions that issues such as fairness, RTT unfairness, friendliness and responsiveness can be studied. In cooperation with other groups we are currently working on creating a suitable infrastructure with multiple servers at multiple sites that would allow such tests.

VIII. Related Work

H-TCP is derived from the basic TCP algorithm by modifying the manner in which a flow's window variable increases between congestion events, and by modifying the manner in which flows backoff at congestion events. Both of these borrow heavily from concepts that have been explored extensively in the TCP-literature.

We have already mentioned that several competing congestion control protocols, BIC-TCP, Scalable TCP, and HS-TCP, move from a linear additive increase law to a non-linear additive increase law. Some of these protocols also employ RTT-scaling to mitigate the extreme effect of RTT-unfairness that arises in large BDP networks. The novelty in our approach is that H-TCP's increase law is designed to maintain symmetry in the manner in which competing flows acquire available bandwidth by increasing the additive increase rate as a function of the time since the last backoff.

Similarly, 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 [12]. In TCP Vegas, and more recently FAST TCP [3], 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. This is 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 we are seek instead to stay within the well tested AIMD paradigm. In the context of wireless networks and error-prone links (so-called "leaky dynamic pipes"), Gerla *et al* [13] consider a TCP variant denoted TCP-Westwood that proposes modifying the AIMD backoff factor based on 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. Conclusions

In this paper we have considered the problem of designing a congestion control protocol that is suitable for deployment in high-speed and long distance networks. The main contribution of our work is to consider this problem from the point of view of a number of basic constraints that must be satisfied by any new protocol. Arising from these considerations, we suggest a new high speed protocol, H-TCP. Results from an experimental testbed are given to demonstrate the efficacy of H-TCP and to compare its performance with competing proposals.

One of our objectives in writing this paper is to initiate a discussion as to what constitutes a good congestion control protocol for large BDP networks. In this paper we focus our attention on a number of obvious features that might be deemed desirable for any new congestion control algorithm. It is reasonable to expect that algorithms should be fair when competing with each other, be friendly toward legacy TCP flows, provide a certain degree of backward compatibility with legacy TCP, should strive to utilise network resources efficiently, should also acquire and release bandwidth quickly in response to changing network conditions, and should be suitable for deployment in a variety of network types (not just large BDP networks). We show that at least one congestion control protocol, H-TCP, can be realised that satisfies all of these objectives in a straightforward manner. The design criteria that we have considered represent a minimum set of requirements that we expect must be satisfied from the point of view of efficient network design. Many other potentially important design criteria have not been considered in this paper. For example, much of our discussion is concerned with the behaviour of long lived flows. Little attention has been paid to the behaviour of short-lived H-TCP flows, to the manner in which high-speed protocols interact with non-responsive flows and active queuing disciplines, and to the manner in which packet bursts are generated in these networks. Our hope is that this paper will initiate debate on the topic as to what exactly are the desirable features of large BDP protocols.

Acknowledgements

This work was supported by Science Foundation Ireland grants 00/PI.1/C067 and 04/IN3/I460.

References

[1] S. Floyd, "High speed TCP for large congestion windows." RFC 3649, Experimental, December 2003.

[2] T. Kelly, "On engineering a stable and scalable TCP variant," tech. rep., Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR.435, 2002.

[3] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance." Caltech CS Report CaltechCSTR:2003:010, 2003.

[4] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks." Proceedings of IEEE INFOCOM, 2004.

[5] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, 1989.

[6] R. Shorten, D. Leith, J. Foy, and R. Kilduff, "Analysis and design of synchronised communication networks," in *Proceedings of 12th Yale Workshop on Adaptive and Learning Systems*, 2003.

[7] R. Shorten, F. Wirth, and D. Leith, "Positive matrices and the internet." IEEE Transactions on Networking, to appear, 2005.

[8] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *In Proceedings of ACM SIGCOMM '04, August /September 2004.*, 2004.

[9] Y. Lee and D. Leith, "BicTCP implementation in linux kernels." Hamilton Institute Technical Report, www.hamilton.ie/net/LinuxBicTCP.pdf, 2004.

[10] D. Leith, "Linux implementation issues in high-speed networks." Hamilton Institute Technical Report, www.hamilton.ie/net/LinuxHighSpeed.pdf, 2003.

[11] R. Shorten, D. Leith, J. Foy, and R. Kilduff, "Analysis and design of synchronised communication networks." Automatica, 2003.

[12] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, pp. 24–35, 1994.

[13] M. Gerla, N. B., M. Sandaidi, M. Valla, and R. Wang, "TCP Westwood with adaptive bandwidth estimation to improve efficiency/friendliness tradeoffs," *Computer Communication Journal*, 2003.