Delay-based AIMD congestion control

D. Leith[1], R.Shorten[1], G.McCullagh[1], J.Heffner[2], L.Dunn[3], F.Baker[3]

***Abstract*— Our interest in the paper is investigating whether it is feasible to make modifications to the TCP congestion control algorithm to achieve greater decoupling between the performance of TCP and the level of buffer provisioning in the network. In this paper we propose a new family of delay-based congestion control algorithms that we refer to as delay-based AIMD.**

I. Introduction

The performance of the standard TCP congestion control algorithm is intimately coupled to the level of buffer provisioning along a path. For example, when buffer provisioning is too low, throughput efficiency falls due to the buffer emptying when TCP flows backoff their congestion windows. Conversely, when buffer provisioning is large, the probing action of the TCP congestion control algorithm seeks to fill the queue thus leading to large queueing delays. Our interest in this paper is investigating whether it is feasible to make modifications to the TCP congestion control algorithm to achieve greater decoupling between the performance of TCP and the level of buffer provisioning in the network. Note that this objective complements the recent discussion in the literature relating to buffer sizing for TCP flows. That is, rather than considering how buffers can be sized to accommodate TCP flows we consider whether TCP can be modified to accommodate network buffers.

In previous work[9], we proposed an approach for effectively decoupling TCP throughput from network buffer size. This was achieved by adapting the congestion window backoff factor in TCP to accommodate the level of buffer provisioning within the network. In this paper we substantially extend this work to consider decoupling of TCP delay from buffer size. Loss-based congestion control algorithms seek to fill the network buffers and so can induce large queueing delays if buffers are large. Note that large buffers are ubiquitous in modern access networks e.g. queueing delays of several seconds are common in DSL links. Large buffers, for example with 250ms or more of buffering, are also still commonplace in high-speed networks. Although the use of smaller buffers is currently the subject of much discussion within the research community, e.g. see [1], [5], the issue remains controversial[4] and it is likely to be some years before such changes could be widely rolled out. As noted above the present work is complementary to this work on buffer sizing, considering end host changes that seek to enhance the flexibility of designers when choosing network buffer sizes.

Consideration of queueing delay inevitably leads to consideration of delay-based congestion control algorithms. Potentially, allocation of network bandwidth between competing sources can be achieved while maintaining low queueing delay (even when network buffers are large, unlike with loss-based algorithms), and with almost full utilisation of network links. High utilisation with low queuing delay is termed "operating at the knee of the curve" and is evidently a desirable property.

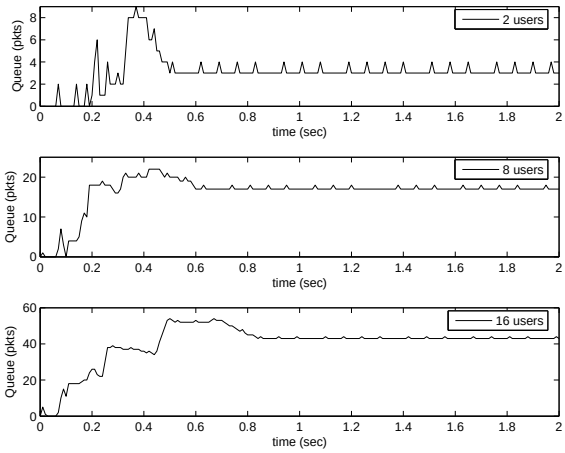

Fig. 1. Queue occupancy for 2, 8, and 16 flows using TCP Vegas. ns simulation, dumbbell topology, RTT 30ms, link rate 5Mbps, 100 packet queue.

Delay-based congestion control algorithms have of course been widely studied, with Vegas[2] and FAST[8], [10] receiving particular attention. However, it is important to note that while Vegas, FAST and related algorithms use delay as a congestion signal, they make no attempt minimize aggregate queuing delay. This is easy to see by considering that the Vegas algorithm seeks to maintain some per flow target number of packets queued in the network. Thus, the number of packets queued scales with the number of flows in the network. This behaviour is illustrated for example in Figure 1. Consequently such protocols do not result in networks operating at the "knee of the curve" in general. TThis observation applies to all Vegas-type and related schemes including TCP-FAST, e.g. see `http://www.cubinlab.ee.mu.oz.au/ns2fasttcp/` for results illustrating this feature of FAST.

In this paper we propose a new family of delay-based congestion control algorithms that we refer to as *delay-based AIMD*. We demonstrate that the delay-based AIMD approach allows algorithms to be realised with the following properties.

1. Networks in which only delay-based flows are deployed operate at the "knee of the curve"; namely; in a low delay, high utilisation regime with zero packet loss.
2. Operation at the "knee" is essentially achieved regardless of the number of network flows and is largely decoupled

[1] Hamilton Institute. [2] Pittsburgh Supercomputing Center. [3] Cisco Systems.
This work was supported by Cisco Systems and by Science Foundation Ireland grants 00/PI.1/C067 and 04/IN3/I460.

from network buffers sizes. It is also robust to perturbations in queue occupancy; in other words, the delay-based flows will co-operate to drain the network queues to a pre-specified threshold whenever only delay-based flows are present in the network.

3. In mixed environments, delay-based and loss-based flows may coexist in a well-defined manner.

II. Delay-based AIMD: Basic Algorithm

To illustrate the basic rationale and features of the delay-based AIMD approach in this section we begin by considering the case of homogeneous networks, i.e. networks where all flows operate the same congestion control algorithm. In Section sec:non we extend consideration to networks with a mix of delay and loss-based flows and propose extensions to the basic delay-based AIMD algorithm to ensure robust co-existence with loss-based flows.

A. Ensuring low queueing delay

Our starting point is to develop an AIMD algorithm that uses delay rather than loss (or ECN) to control network congestion. Standard TCP employs an *Additive-Increase Multiplicative-Decrease* (AIMD) [3] strategy during its congestion avoidance mode. AIMD congestion control can be implemented using signals other than packet loss as a congestion indicator. The basic idea here is that by backing off when queueing delay exceeds some threshold, we can avoid filling the queue (maintain low queueing delay) while staying within the well-established AIMD framework. Specifically, we consider the following delay-based AIMD algorithm:

$$cwnd \leftarrow \begin{cases} cwnd + \alpha/cwnd, & on\ each\ ACK \\ \beta cwnd, & if\ \tau \geq \tau_0\ \&\ cwnd > w_0 \\ \beta cwnd, & if\ packet\ loss \end{cases} \quad (1)$$

where τ is the observed queueing delay, $\tau_0 > 0$ is a delay threshold that triggers delay-based backoff, and w_0 specifies a $cwnd$ threshold above which the delay-based action is activated (this may be helpful, for example, to ensure that there are sufficient packets in flight to provide a reliable estimate of queueing delay). The queueing delay τ is estimated as $sRTT(t) - RTT_{min}$ where RTT_{min} is the minimum observed packet round-trip time and $sRTT(t)$ is a smoothed estimate of the current round-trip time. Loss induced backoffs are retained as part of the algorithm to accommodate situations where, for example, the buffers are sized such that maximum queueing delay is less than τ_0 or where the bandwidth-delay product is less than w_0. Since it continues to employ an AIMD strategy, the delay-based algorithm inherits the usual fairness and convergence properties of AIMD.

The impact of this change on the AIMD operation is illustrated in Figure 2. It can be seen that although the buffer is sized at 400 packets, the flow $cwnd$ now backs off before the queue is full. In this example we can also see that following the first backoff where $cwnd$ is reduced by half, the queue empties for a significant period of time – this is to be expected as the delay-based algorithm backs off $cwnd$ before the queue is full. High utilisation be maintained by adjusting the backoff factor appropriately – this is illustrated in subsequent backoff events in Figure 2 and is discussed in detail in the Section II-C.

Congestion window (packets)	Congestion epoch duration (s)
100	1.1
1000	3.1
2000	4.3
5000	6.6
10000	9.2
20000	12.8
50000	19.4

TABLE I

CONGESTION EPOCH DURATION VERSUS BANDWIDTH-DELAY PRODUCT. VALUES TAKEN FROM H-TCP INTERNET DRAFT[7].

B. Scalability to high bandwidth-delay product paths

It is important to emphasise that the choice of additive increase function α used in the delay-based algorithm is flexible – any additive increase algorithm can be extended as shown here to include backoff on queueing delay as well as loss. In the rest of this paper we adopt the increase function used in the H-TCP loss-based algorithm[6], [7]. This increase function ensures that performance scales well to high bandwidth-delay product paths while maintaining backward compatibility in low-speed environments and has already been the subject of extensive experimental testing, see for example [11] and references therein. The H-TCP increase function is

$$\alpha = \min[1, 1 + 10(\Delta - 1) + 0.5(\Delta - 1)^2] \quad (2)$$

That is, the AIMD increase rate α is a function of Δ, the elapsed time since the last backoff event. This yields a congestion epoch duration that scales as shown in Table I while leaving other properties (fairness, convergence rate etc) of TCP essentially unchanged[6], [11].

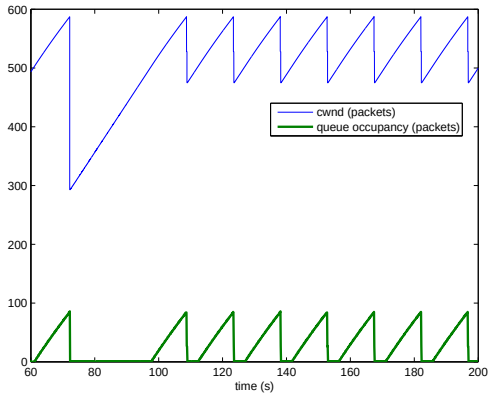

Fig. 2. Illustrating adjusting backoff factor to maintain high link utlisation. (ns simulation, delay 120ms, link rate 50Mbps, 400 packet queue, τ_0 20ms).

C. Ensuring high utilisation

Low queueing delay requires that we select τ_0 to be small. However, this can lead to low link utilisation if the flows

respond too aggressively to congestion. Fortunately, following [9] it is straightforward to maintain a high level of utilisation by adjusting the AIMD backoff factor to reflect the queuing delay when a backoff occurs. Namely, we set

$$\beta = RTT_{min}/RTT(t) \tag{3}$$

We can understand the effect of this choice of backoff factor in more detail as follows. Consider a single bottleneck link with n flows (the following argument also extends directly to multiple bottleneck situations). Before a backoff of the congestion window, the data throughput through the bottleneck link is given by $R^- = \sum_{i=1}^{n} w_i/RTT_i(t) = B$ where B is the link rate in packets per second, w_i is the congestion window of flow i immediately before backoff and RTT_i is the RTT of flow i. Following a backoff the data throughput is given by $R^+ = \sum_{i=1}^{n} \beta_i w_i/RTT_{min,i}$, where β_i is the backoff factor of flow i and RTT_{min} is the propagation delay . Selecting β_i according to (3), it follows immediately that $R^+ = R^- = B$ and link utilisation is 100%.

The effect of this AIMD modification is illustrated in Figure 2. In this example the queue empties for a substantial period following backoff by a factor of 0.5 (the first backoff event in the figure) with an associated reduction in link utilisation. Once the flow adjusts its backoff factor to the effective level of buffer provisioning at backoff we can see that the queue now just empties following a backoff event and the link continues to operate close to capacity as desired.

D. Draining the queue – robust delay-based operation

Like other delay-based algorithms that have been proposed, the delay-based AIMD algorithm requires an estimate of the queueing delay τ. However, unlike the majority of these algorithms, the delay-based AIMD algorithm can be designed so that flows will always act in a collaborative manner to drain the network buffer irrespective of the initial level of buffer occupancy. This latter feature is important for a number of reasons. Firstly, loss-based flows, or in some circumstances non-elastic traffic, will act to fill network buffers. A return to operation at the "knee" requires that there is some mechanism for network buffers to drain after such traffic has switched off. Secondly, draining the buffer allows the propagation delay RTT_{min} to be observed by senders, which is key to accurate estimation of queueing delay.

Fortunately, this basic property can be easily realised by ensuring that each flow responds in a way that seeks to reduce the minimum RTT that it has seen up to this point. Specifically, we consider estimating the queueing delay using $\tau = sRTT(t) - RTT_{min}$. Evidently, if the queues never empty, the flow never observes the path propagation delay and RTT_{min} will be an overestimate of the true propagation delay. When RTT_{min} is larger than the true propagation delay, the queueing delay is consistently underestimated by a flow and this may lead to operation with a standing queue (operation away from the knee of the curve).

To help gain some insight, consider for the moment a network with a single flow. The flow will increase its congestion window until the estimated queueing delay $\tau = RTT(t) - RTT_{min}$ exceeds threshold τ_0, and then backoff. The actual queueing delay at backoff will be $\tau_0 + RTT_{min} - T_{min,i}$, where $RTT_{min} - T_{min}$ is the estimation error in propagation delay. The backoff factor (3) decreases the number of packets in flight so as just empty estimated queue i.e. such that the round-trip time falls to RTT_{min}. If we now modify the backoff factor to be

$$\beta = \delta RTT_{min}/RTT(t) \tag{4}$$

with $0 < \delta < 1$, then in effect we can use the multiplicative decrease action to probe the network to discover whether an RTT *below* RTT_{min} is possible. This is illustrated in Figure 3 for the case of a single flow. Figure 4 gives an example of the convergence of RTT_{min} in a network with multiple flows.

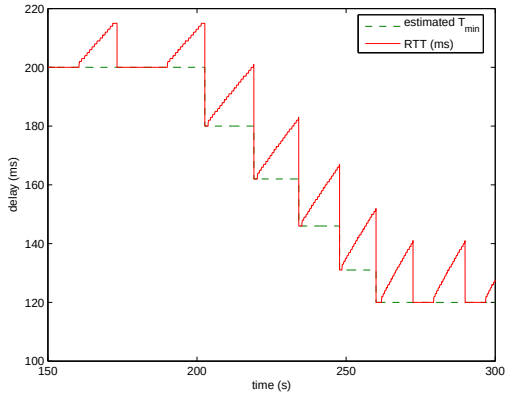

Fig. 3. Example of ratchetting down of base RTT estimate RTT_{min} until it equals the actual propagation delay. The path propagation delay is 120ms while RTT_{min} is manually configured to an initial value of 200ms for illustrative purposes. (ns simulation, propagation delay 120ms, bandwidth 50Mbps, 400 packet queue, τ_0 20ms).

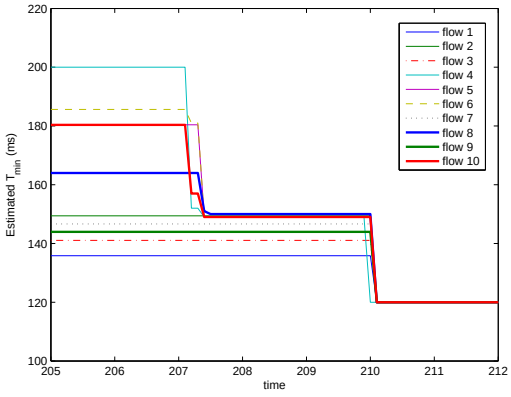

Fig. 4. Example of ratchetting down of base RTT estimate RTT_{min} in a network with 10 delay-based flows. The path propagation delay is 120ms while RTT_{min} for each flow is initially set to a random value uniformly distributed between 120ms-240ms for illustrative purposes. (ns simulation, 10 flows, propagation delay 120ms, bandwidth 50Mbps, 400 packet queue, τ_0 20ms, $\delta = 0.75$).

III. Co-existence with legacy loss-based TCP

Delay-based flows compete poorly with loss based flows as they experience an excessive number of backoffs as loss-based flows fill the network queues. This problem can be addressed as follows.

First, we note that we have some freedom in the selection of the delay threshold τ_0. We exploit this and choose τ_0 to be proportional to the recent level of queue occupancy. In this way, the threshold automatically adjusts upwards when loss-based flows are present that fill the queue, thus enabling delay-based flows to increase their congestion window. Specifically, we choose τ_0 according to

$$\tau_0 = (1-\gamma)\bar{\tau}_0 + \gamma(RTT_{max} - RTT_{min}) \qquad (5)$$

with $0 < \gamma < 1$ and where RTT_{max} is a quantity that tracks the maximum observed RTT and decays towards RTT_{min} during periods when the current RTT is below RTT_{max}. $RTT_{max} - RTT_{min}$ is an estimate of the recent queue occupancy and τ_0 is selected to be a convex combination of the baseline value $\bar{\tau}_0$ and $RTT_{max} - RTT_{min}$. When $\gamma = 0$ we recover the previous delay-based AIMD algorithm. When $\gamma > 0$, delay-based flows are able to increase their congestion window even when the network queues are persistently backlogged due to the action of loss-based flows.

Second, we modify the delay-based AIMD algorithm to (i) perform additive increase of the congestion window only when the queueing delay is below threshold τ_0 – the effect is to disable the AIMD probing action once the queueing delay rises above τ_0, and (ii) to multiplicatively decrease the congestion window when the queueing delay is at or above τ_0 and the time since the last backoff is greater than a threshold Δ_0 – the effect is to limit the number of delay induced (as opposed to loss induced) backoffs in a given time interval.

An illustrative congestion window time history is shown in Figure 5. It can be seen that these modifications amount to inserting a flat section, where the congestion window is constant, into the usual sawtooth waveform. The time between backoffs, and so the length of this flat section, is determined by the value of Δ_0. The flat section is essentially an idling period that does not alter the dynamics of a network of flows other than to extend the duration of the congestion epochs. When delay-based flows are competing against loss-based ones, however, this idle period constrains the rate at which delay-based flows backoff in response to the level of queueing delay and can therefore be used to adjust how bandwidth is divided up between delay and loss-based flows.

Combining these changes yields the following modified AIMD algorithm: on each ACK,

$$cwnd \leftarrow \begin{cases} cwnd + \alpha/cwnd, & if\ cwnd \le w_0\ or\ \tau \le \tau_0 \\ \beta cwnd, & if\ cwnd > w_0\ \&\ \tau \ge \tau_0 \\ & \&\ \Delta > \Delta_0 \\ \beta cwnd, & if\ packet\ loss \end{cases} \qquad (6)$$

where Δ is the elapsed time since the last backoff. One option is to take Δ_0 to be proportional to $RTT_{max} - RTT_{min}$ so that we recover the original delay-based AIMD algorithm when the queue backlog remains low and $RTT_{max} - RTT_{min}$ small.

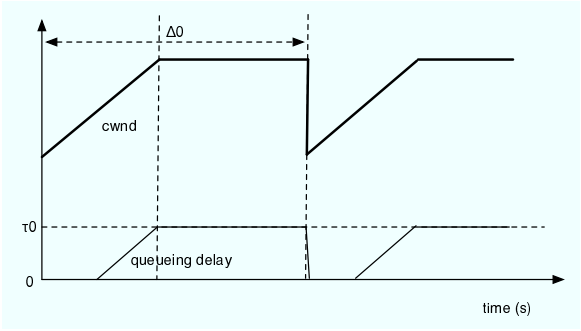

Fig. 5. Example congestion window time history using modified AIMD algorithm.

The impact of these changes on fairness is illustrated in Figure 6.

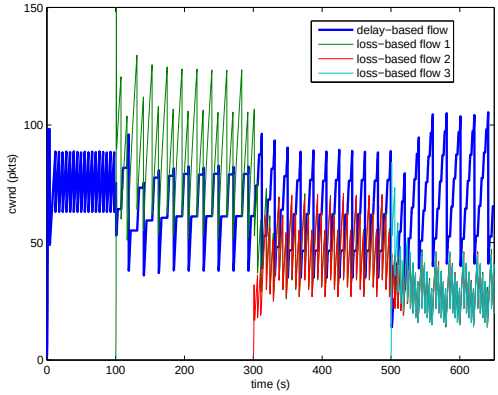

Fig. 6. Example of heterogeneous network with a mix of delay and loss-based flows illustrating the impact on the delay-based flow's performance of the modified AIMD algorithm. (ns simulation, 1 delay-based flow, 3 loss-based flows, propagation delay 100ms, bandwidth 10Mbps, 100 packet queue).

IV. Delay-based slow-start

With the goal of minimizing queuing delay and loss, slow-start poses a significant problem. Its exponential increase, where $cwnd$ is doubled each RTT, can create large bursts of packets that in turn cause large delay spikes and many losses when $cwnd$ eventually overshoots the botteneck queue. We can apply the same delay backoff threshold τ_0 as used in the delay-based AIMD algorithm to trigger an exit from slow-start when queueing delay rises. This will keep the queue from filling. However, due to the bursty nature of slow-start, with associated spikes in queueing delay, it causes slow-start to exit sooner than it should.

In order to continue a fast increase rate, but avoid the queuing spikes of slow-start, we propose a mechanism based on Limited Slow-Start [RFC3742]. This uses a parameter, $max\_ssthresh$, which controls the maximum bottleneck queue occupancy the flow will contribute due to congestion window increase. To acheive this it bounds the increase of $cwnd$ to no more than $max\_ssthresh/2$ per round-trip time.

This algorithm meshes nicely with the delay-based AIMD algorithm described above, as they both seek to keep queue

occupancy below a set point. A difficulty in using limited slow-start is selecting an appropriate $max\_ssthresh$. When used in conjunction with delay-based AIMD, $max\_ssthresh$ should be set so that a bottleneck queue occupancy of $max\_ssthresh$ corresponds to a queue delay of τ_0.

To find this conversion factor, we use the initial slow-start phase to estimate the bottleneck rate. An ack-clocked slow-start sends at twice the rate of the ack clock, up to twice the bottleneck rate. This results in transient queue increases of size $cwnd(t)/2$ where $cwnd(t)$ is the congestion window at time t. The drain time of this queue can be observed by the returning ACKs. When $RTT > RTT_{max}$, the queue's drain time can be estimated as $RTT_{max} - RTT_{min}$. The queue's size is $cwnd(t - RTT)/2$, since the ACKs are received one RTT after the data is sent, or $cwnd(t)/4$, since the window doubles each RTT. This estimation of both queue length over drain time gives us bottleneck rate we need to convert τ_0 to $max\_ssthresh$. Specifically, when $RTT > RTT_{max}$:

$$max\_ssthresh \leftarrow \left(\frac{cwnd}{4}\right)\left(\frac{\tau_0}{RTT_{max} - RTT_{min}}\right) \quad (7)$$

Comment : A significant advantage of using the parameter τ_0 (units of time) over $max\_ssthresh$ (units of bytes) is that the behavior becomes scale independent of data rate. Limited slow-start takes $O(cwnd/max\_ssthresh)$ RTTs to reach a given window, but this delay-based version takes $O(RTT/\tau_0)$ RTTs.

This approach is substantially different from the experimental slow-start (Vegas*) proposed in [2], which measures dispersion of the initial four-segment window to estimate the bottleneck rate, then uses a pacing mechanism to smooth out the slow-start. Both methods effectively limit delay spikes, but we found the limited slow-start approach simpler as it does not require an additional timer mechanism.

V. Complete algorithm

While the foregoing discussion is quite complex, the complete delay-based algorithm is itself very simple. The new delay-based AIMD algorithm is shown in its entirety in Algorithm 1.

Algorithm 1 : Pseudo code of complete delay-based AIMD algorithm with limited slow-start

On each ACK:
$RTT_{min} = \min(RTT, RTT_{min})$
$RTT_{max} = \max(RTT, RTT_{max} - a \times RTT/cwnd)$
$\tau = RTT - RTT_{min}$ // estimate queueing delay
$\beta = \min(\delta RTT_{min}/RTT, 0.9)$
if $cwnd \leq ssthresh$ **then**
  if $cwnd \leq \hat{s}sthresh$ **then**
    if $\tau = \hat{T}_{max}$ **then**
      $max\_ssthresh = (cwnd/4) \times \tau_0/(RTT_{max} - RTT_{min})$
    end if
    if $cwnd \leq max\_ssthresh$ **then**
      $cwnd+ = MSS$
    else
      $cwnd+ = max\_ssthresh/(2 \times cwnd)$
    end if
  end if
else
  if $cwnd \leq w_0$ or $\tau \leq (1-\gamma)\bar{\tau}_0 + \gamma(RTT_{max} - RTT_{min})$ **then**
    $cwnd+ = 2(1-\beta)\alpha(\Delta)/cwnd$
  end if
end if
if $cwnd > w_0$ & $\tau \geq \bar{\tau}_0$ & now − time_of_last_backoff $> \Delta_0$ **then**
  $cwnd = \beta \times cwnd$
  $ssthresh = cwnd$
  time_of_last_backoff = now
end if

	Description
CPU	Intel Xeon CPU 2.80GHz
Memory	256 Mbytes
Motherboard	Dell PowerEdge 1600SC
Kernel	Linux 2.6.6
txqueuelen	1,000
max_backlog	300
NIC	Intel 82540EM
NIC Driver	e1000 5.2.39-k2
TX & RX Descriptors	4096

TABLE II
Hardware and Software Configuration.

VI. Experimental Results

We have implemented the delay-based AIMD algorithm in Linux 2.6.17 and in this section we present initial experimental results exploring the performance of the algorithm.

A. Test setup

Experiments were carried out using a high-speed testbed. The testbed consists of commodity PCs connected to gigabit switches to form the branches of a dumbbell topology. All sender and receiver machines used in the tests have identical hardware and software configurations as shown in Table II and are connected to the switches at 1Gb/sec. The router, running the FreeBSD dummynet software, can be configured with various bottleneck queue-sizes, capacities and round trip propagation delays to emulate a range network conditions. Flows are injected into the testbed using `iperf`.

B. Operation at the "knee"

Figure 7 shows the measured link utilisation and mean link delay (measured using pings) versus number of flows for the delay-based AIMD algorithm using the H-TCP increase function. It can be seen that the link utilisation is close to the link capacity regardless of the number of flows. With the delay-based algorithm the link delay remains consistently low (close to the propagation delay of 250ms) regardless of the number of competing flows sharing the link, with the mean queueing delay remaining less than 30ms at all times. In these tests we observed a packet loss rate of zero i.e. no losses whatsoever, even with 128 flows sharing the link.

C. Fairness and Convergence

Since the delay-based algorithm remains within the AIMD paradigm, it inherits many of the properties of standard TCP. In particular, the unfairness between flows with different RTTs is similar to that for standard TCP although we do not present measurements demonstrating this here due to space restrictions. The delay-based algorithm also inherits similar convergence properties as other AIMD algorithms. Conver-

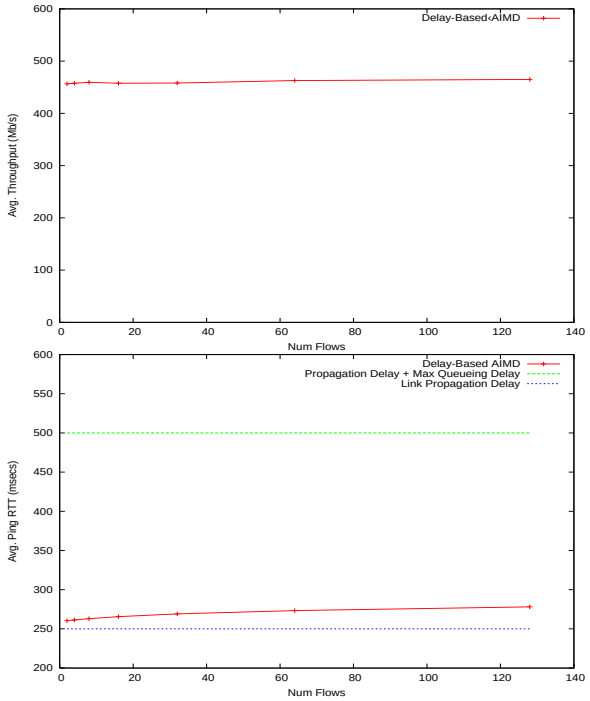

Fig. 7. Measured link utilisation and delay with loss-based H-TCP and the new delay-based algorithm (using τ_0=50ms). 500Mbps link rate, 250ms RTT, bandwidth-delay product of buffering.

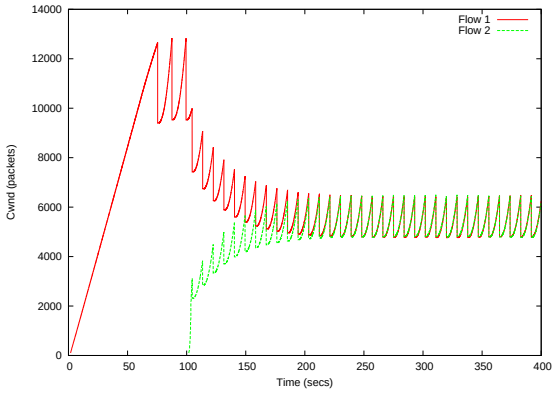

Fig. 8. Convergence of delay-based algorithm following startup of a second flow. 500Mbps link rate, 250ms RTT, 250ms of buffering.

gence 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 β_{max} in the network, with the 90% rise time measured in congestion epochs bounded by $\log 0.1 / \log \beta_{max}$ (yielding a rise time of 3 congestion epochs for a backoff factor of 0.5 and 7 congestion epochs for a backoff factor of 0.75). Note that the adaptation of the AIMD backoff factor to main high throughput in the delay-based AIMD algorithm, which will generally lead to a backoff factor greater than 0.5, can therefore be expected to impact on the convergence rate. This is illustrated, for example, in Figure 8. In this example we have disabled slow-start so as to highlight the congestion avoidance convergence behaviour. Here, the backoff factor adapts to a value of approximately 0.75 and it can be seen that following startup of the second flow at 100s the network converges close to equilibrium in approximately 7 congestion epoch as expected. It is important to note that while the number of congestion epochs for the network to converge is higher when the backoff factor is greater than the standard TCP value of 0.5, the duration of each epoch is shorter with the delay-based scheme since it avoids filling the queue. Overall, the impact of the adjustment of backoff factor in the AIMD algorithm is therefore quite small.

VII. Scope of the Paper

In this paper we present the new delay-based AIMD algorithm and preliminary experimental measurements of performance. Space restrictions naturally limit the number of results that we can show. As a result, we restrict consideration to single bottleneck links and do not present results showing operation over multiple bottleneck links (including queueing on reverse path links). Also, due to space restrictions we do not discuss signal processing issues relating to the estimation of delay. That is not to say that these issues are not important, but analysis and results on these topics will be the subject of future publications.

VIII. Conclusions

In this paper we propose a new family of delay-based congestion control algorithms that we refer to as delay-based AIMD. This class of algorithms supports low-delay, high-throughput operation essentially regardless of buffer provisioning within the network and of the number of flows sharing a link. We have implemented in Linux a delay-based AIMD algorithm based on the loss-based H-TCP algorithm and present initial experimental results illustrating the effectiveness of the proposed approach.

References

[1] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proc. ACM SIGCOMM 2004*, 2004.

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

[3] D. Chiu and R. Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks*, 17:pp. 1–14, 1989.

[4] A. Dhamdhere and C. Dovrolis. Open issues in router buffer sizing. *Computer Communications Review*, 36:pp. 87–92, 2006.

[5] A. Dhamdhere, H. Jiang, and C. Dovrolis. Buffer sizing for congested internet links. In *Proc. INFOCOM Miami, FL, 2005*, 2004.

[6] D.J.Leith and R.N.Shorten. H-TCP protocol for high-speed long-distance networks. In *Proc. 2nd Workshop on Protocols for Fast Long Distance Networks. Argonne, Canada, 2004*, 2004.

[7] D.J.Leith and R.N.Shorten. H-TCP protocol for high-speed long-distance networks. In *Internet draft draft-leith-tcp-htcp-02.txt*, 2006.

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

[9] R. Shorten and D. Leith. On queue provisioning, network efficiency and the delay-bandwidth product. IEEE Transactions on Networking, to appear, 2006.

[10] J. Wang, D. X. Wei, and S. H. Low. Modelling and stability of FAST TCP. In *Proceedings of INFOCOM*, March 2005.

[11] Y.Li, R. Shorten, and D. Leith. Experimental evaluation of tcp protocols for high-speed networks. IEEE Transactions on Networking, to appear, 2006.