

# Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms

Douglas J. Leith\*, Lachlan L. H. Andrew†, Tom Quetchenbach†, Robert N. Shorten\*, Kfir Lavi\*  
 \*Hamilton Institute, Ireland †Caltech, Pasadena, CA, USA

**Abstract**—We present initial experimental results for TCP Illinois and Compound TCP. These tests are for relatively simple scenarios yet they are sufficient to highlight several interesting issues. We observe that both TCP Illinois and Compound TCP can exhibit poor scaling behaviour as path BDP increases. As a result, link utilisation can be low and network responsiveness can become sluggish as BDP increases. We also document a number of important implementation issues observed during our tests.

## I. INTRODUCTION

TCP Reno is well known to scale poorly to large bitrate-delay product (BDP) networks, even with extensions such as newReno, SACK and ECN. This is because it only increases its window by one packet per round trip time (RTT) and decreases it by half when a loss is detected. Most proposed solutions [1], [2] alter the rate of increase and/or decrease based purely on loss information. Others [3] adjust the window primarily on the estimated queueing delay. Several promising proposals [4], [5], [6], [7] seek to combine both. This paper empirically evaluates the performance of two such algorithms: Compound TCP [4] and TCP Illinois [5]. To our knowledge this is the first reported experimental evaluation both of TCP Illinois and of the latest version of Compound TCP (in Vista SP1). We focus on simple scenarios which evaluate basic properties such as scaling behaviour, fairness and friendliness as well as situations which are known to cause difficulties for delay-based algorithms. Use of relatively simple scenarios allows the mechanisms to be understood, as a precursor to defining more complex “typical usage” tests to evaluate their performance in the wider Internet.

## II. PRINCIPLES OF OPERATION

TCP Illinois [5] uses an AIMD algorithm but adjusts the increase and decrease parameters based on estimates of queueing delay and buffer size. When no queueing delay is detected, the rate of additive increase (AI) is fixed at 10 packets per RTT; as the estimated queueing delay rises the AI rate is gradually reduced to 0.3 packets per RTT when the estimated queueing delay is maximal (network buffers are estimated to be full). On loss, the congestion window  $cwnd$  is decreased as  $cwnd \leftarrow (1 - \delta)cwnd$ . If the RTT is close to the maximum observed value then the loss is deemed a buffer overflow and  $\delta \approx 1/2$ , while  $\delta$  decreases to  $1/8$  as the RTT gets smaller (as loss is then taken as packet corruption).

Compound TCP [4] maintains a Reno-like AIMD window  $wnd$ , and a separate delay-based window,  $dwnd$ . The congestion window used is the sum of the two. The AIMD window

$wnd$  behaves as per Reno, i.e., increases by one packet per RTT and backs off by  $1/2$  on loss. The delay-based window  $dwnd$  increases rapidly when the estimated queue occupancy is small, using a rule based on HS-TCP [8]. When the estimated number of packets that a flow has queued exceeds a target value  $\gamma$ ,  $dwnd$  is reduced by the estimated number of packets in the queue. On loss,  $dwnd$  also backs off by  $1/2$ . The versions which we use employ TUBE [4] to adapt  $\gamma$  to the available buffer size.

Note that we refer to the applied congestion window (sum of  $wnd$  and  $dwnd$ ) as  $cwnd$ , in line with standard notation although this differs from the terminology in [4].

## III. EXPERIMENTAL SETUP

These experiments were performed on Caltech’s WAN-in-Lab [9] and on the Hamilton Institute testbed based on dummynet, as specified in the text.

WAN-in-Lab servers had 2.4GHz Opteron 250 CPUs with 1MByte cache, and Neterion s2io 10Gbit/s Ethernet cards using the 2.0.25.1 driver, a `txqueue` length of 1000 and `netdev_max_backlog` of 300. TCP segmentation offloading (TSO), interrupt moderation and SACK were enabled. The Hamilton Institute testbed used PE860 servers with Intel Xeon 3GHz 2MB cache CPUs and Intel Pro/1000 GiGE cards.

Tests with TCP Illinois used a Linux 2.6.23.1 kernel with a bug in the Illinois code fixed. Compound TCP tests were performed using both the Windows Vista Service Pack 1 (SP1, build 6001) version of CTCP and a re-implementation for Linux [10], based on the Internet Draft [11]. By working with both implementations our aim is to help verify whether the draft provides a complete description of the official implementation. With Vista,  $cwnd$  time histories are reconstructed from `tcpdump` data while with Linux `tcp_probe` instrumentation is used.

Some tests, including all Vista tests, used dummynet (with 0.1ms scheduler granularity) instead of the fibre delays of WAN-in-Lab. WAN-in-Lab seeks to avoid artefacts, such as packet bursts associated with scheduler granularity, which may be particularly important when studying algorithms which rely on delay. Tests which were performed on both WAN-in-Lab and dummynet can be used to quantify the impact of such artefacts, or lack thereof.

All throughput values are one second averages.

## IV. SCALING BEHAVIOUR WITH BDP

The basic motivation behind the proposed changes to the TCP congestion control algorithm considered here is that

the performance of the current Reno congestion avoidance algorithm scales poorly with path BDP. The Reno congestion avoidance algorithm increases  $cwnd$  by one packet per RTT until network buffers fill and packet loss occurs, at which point  $cwnd$  is reduced by  $1/2$ . The time between congestion-related loss events is referred to here as the congestion epoch duration. With Reno, the congestion epoch duration of a single flow increases linearly in proportion to the path BDP and congestion epoch durations of 30 minutes or longer are common on modern high-speed links.

It is known that this can lead to low link utilisation – since a flow increases  $cwnd$  by only one packet per RTT following loss, losses induced by slow-starting web traffic etc can prevent the  $cwnd$  of a flow from growing sufficiently large to fully utilise a link. As a result link utilisations of only 10-20% are not uncommon with Reno flows on high BDP paths in some situations.

In addition to low link utilisation, long congestion epoch durations are known to lead to sluggish responsiveness to changes in network conditions such as the startup of new flows. This arises because the convergence time is directly related to the congestion epoch duration, with a network of long-lived Reno flows taking roughly four congestion epochs to converge to within 95% of steady state [12]. We note that slow convergence can translate into prolonged unfairness between competing flows, either when a new flow starts or when some flows observe a loss event that others miss.

In the rest of this section we investigate the corresponding link utilisation and convergence rate behaviour of the TCP Illinois and Compound TCP congestion control algorithms.

#### A. Link utilisation

1) *Response function*: A basic requirement is that a single flow should be able to achieve high link utilisation without requiring unrealistically low packet loss rates. We can capture this requirement via the response function, i.e., a plot of throughput (or  $cwnd$ ) vs loss rate for a single flow and randomly generated losses. Recent proposals for changes to TCP all modify the response function to be more aggressive than Reno.

TCP Illinois increases  $cwnd$  by 10 packets per RTT when the queueing delay is estimated to be low, and by an amount decreasing to 0.3 packets as the queueing delay rises. Thus the response function of TCP Illinois lies between that of a Reno flow with increase rate 0.3 packets per RTT and one with increase rate of 10 packets per RTT.

Note that the increase rate is fixed at a maximum of 10 packets per RTT. Since Ethernet increases its bitrate by a factor of 10 each generation, it follows that TCP Illinois essentially buys us one extra generation; beyond that, similar scaling issues to Reno will inevitably manifest themselves.

Compound TCP uses an additive increase rate based on that of HS-TCP when the queueing delay is small. The increase rate therefore increases with congestion window and is not capped at a small fixed value.

To illustrate this difference, Fig. 1 shows the response of TCP flow on a 400Mbps link to a 5s burst of UDP traffic at

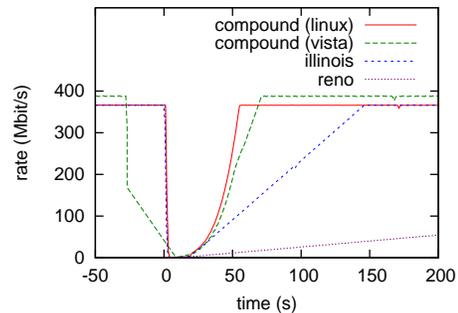


Fig. 1. Response to 5s interruption by UDP traffic at 100s. 400Mbps link, 200ms RTT,  $1 \times BDP$  buffer.

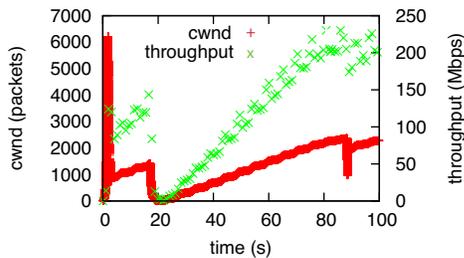
time 0, which forces the  $cwnd$  to almost 0. Since the  $cwnd$  is reduced due to loss, recovery is via congestion avoidance rather than slow start. TCP Illinois takes approximately 150s to fully utilise the link following the disturbance, while the Linux implementation of Compound TCP takes about 60s. In order to reduce its window fully the Vista implementation of Compound TCP, 20s cross traffic was needed and the recovery started about 30s after the onset of cross traffic. Once recovery started, the behaviour was similar to that of the Linux re-implementation.

#### 2) Impact on response function of reverse path queueing:

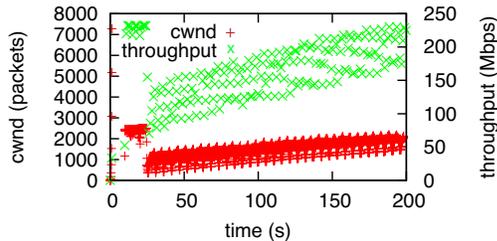
The aggressiveness, and therefore the response functions, of TCP Illinois and Compound TCP are adjusted based on measured queueing delay. Errors in estimating queueing delay can therefore be expected to impact link utilisation. To explore this we briefly investigate the link utilisation of TCP Illinois and Compound TCP in the presence of reverse path traffic.

Fig. 2(a) shows measurements for a 250Mbps link with a single forward path TCP Illinois flow. Reverse path traffic starting at 25s is on-off (5s on, 1s off) with a window capped at 0.5 MByte, giving a maximum of approximately 40Mbps, and an observed throughput on the reverse path of 20Mbps. No losses were measured on the reverse path during the experiment. TCP Illinois takes around 60s to fill the link following a loss at time 25s. Queueing of acks due to the reverse path traffic causes TCP Illinois to incorrectly infer queue buildup and thus to adjust its rate of increase to around 3 packets per RTT, rather than 10 packets per RTT. However, since TCP Illinois does not backoff  $cwnd$  except on loss, it recovers faster than Reno would.

The behaviour with Compound TCP is shown in Figure 2(b). It can be seen that the impact of reverse path queueing is substantial. Following a loss at time 25s, the Compound TCP flow takes around 200s to fill the link, i.e., the response function and link utilisation performance is similar to that of Reno. The reverse queueing delay causes the Compound flow to consistently backoff  $dwnd$  early. As a result, the reverse path traffic prevents the Compound TCP  $dwnd$  from growing and so Compound TCP reverts to Reno-like behaviour. Thus link utilisation becomes poor as path BDP increases. Note that the volume of reverse path traffic is less than 10% of link capacity. However, the frequent slow-starting is sufficient to create burstiness and queueing on the reverse path.



(a) TCP Illinois



(b) Compound TCP

Fig. 2. Behaviour in presence of reverse path traffic. Single forward path flow, on-off reverse path flow with window capped at 0.5 MByte. Link 250 Mbps, 100 ms RTT,  $1 \times \text{BDP}$  buffer.

### B. Convergence time

Qualitatively, the convergence time of an algorithm is the time it takes for a new flow, after entering congestion avoidance mode, to obtain its average fair share of the capacity. For a formal mathematical discussion, see [12].

Note that, while hybrid loss/delay algorithms adjust the congestion window increase and decrease parameters based on delay, fundamentally these algorithms still operate in terms of congestion epochs. In particular, from [5] we expect the convergence time of TCP Illinois to be proportional to the congestion epoch duration, and similarly with Compound TCP.

Fig. 3 plots measured congestion window time histories for a single TCP Illinois flow on a 10 Mbps and a 250 Mbps link. The congestion epoch duration is already rather long (around 600 s) at 250 Mbps. TCP Illinois reverts to a less aggressive AIMD increase rate on detecting queueing delay, to as little as 0.3 packets per RTT. Because the additive increase rate can be a third that of Reno, it takes up to three times as long to fill a buffer. This can be seen in Fig. 3 where TCP Illinois takes 600 seconds to increase cwnd by 2000 packets on a 100 ms RTT link, compared with about 200 s for Reno which increases by one packet per RTT. Since the convergence time is proportional to congestion epoch duration, convergence can be slow following a change in network conditions. This is illustrated, for example, in Fig. 4.

The congestion epoch duration with Compound TCP is determined entirely by the loss-based component of the algorithm, provided that the buffer is large enough that dwnd decreases to zero before loss. That is, the congestion epoch duration is generally identical to that of Reno and so scales linearly with BDP. Figure 5 plots the window size for the Vista implementation of Compound TCP when a new flow starts. This data, and results from other experiments, indicates that the convergence time scales roughly with the congestion epoch duration and is similar to that of Reno. That is, the convergence time scales linearly with BDP and thus the network can exhibit

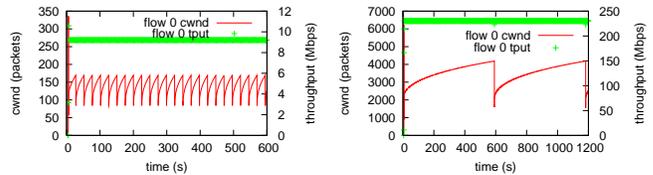
(a) 10 Mbps, 100 ms,  $1 \times \text{BDP}$ (b) 250 Mbps, 100 ms,  $1 \times \text{BDP}$ 

Fig. 3. TCP Illinois congestion epoch duration vs BDP (Dummysnet). Cases with smaller buffers are considered in Section VI.

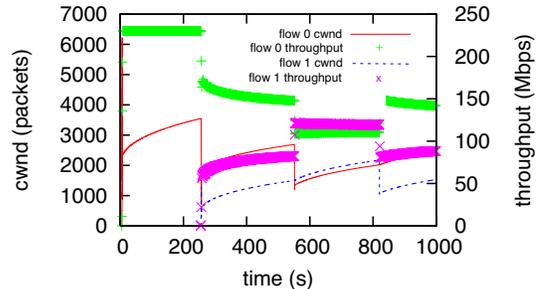


Fig. 4. TCP Illinois convergence time. Best case example where new flow does not backoff at first congestion event, yet still takes around 300 s to converge. On missing drops, sustained excursions from fairness occur. 250 Mbps link, 100 ms RTT,  $1 \times \text{BDP}$  buffer. (Dummysnet)

poor responsiveness at larger BDPs.

When dwnd is non-zero for a significant fraction of each congestion epoch, its dynamics also affect the convergence behaviour. In Figure 6, two flows share a link with a relatively small buffer. Both flows exit slow start while the loss-based wnd is still small, since the buffer cannot accommodate the bursts resulting from the doubling of the window each RTT during slow start. In Figure 6 it takes around 500 s for the loss-based wnd to fill the 2500 packet pipe. During this time, convergence is dominated by dwnd. Figure 6 (and other experimental results) show that the convergence time of the dwnd dynamics can be quite sluggish – the flows have not converged to fairness after 300 s.

## V. RTT FAIRNESS

Loss-based high-speed algorithms often introduce RTT-unfairness, unless this is explicitly counteracted [1]. In contrast, fairness between Vegas flows is generally unaffected by differences in RTT, which has led to claims that hybrid approaches may be more RTT friendly than Reno [4].

Figure 7 shows the ratio of the throughputs achieved by a flow with 120 ms (base)RTT and a flow with shorter RTT, sharing a 100 Mbit/s link with 10% cross traffic, half of which had RTT 6 ms and half had RTT 120 ms. The cross traffic consisted of a Poisson arrival process of flows with Pareto distributed file sizes, with mean 25 000 bytes and shape parameter  $\alpha = 1.2$ , and was included to avoid phase effects.

Results are shown when the buffers are sized at 120 ms and 16 ms ( $\text{BDP}$  and  $0.13\text{BDP}$  for the 120 ms flow).

When the buffer is 120 ms, Compound TCP and TCP Illinois exhibit similar RTT unfairness to Reno. This is to be expected, since their dynamics approach that of Reno when

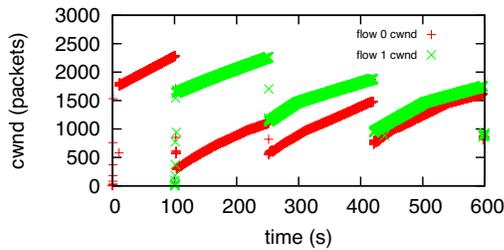


Fig. 5. Compound TCP convergence time following startup of a second flow at time 100 s. 200 Mbps link, 100 ms RTT,  $1 \times \text{BDP}$  buffer. (Vista, dummynet)

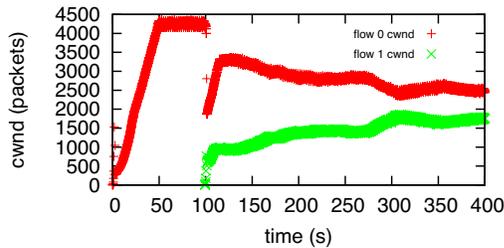


Fig. 6. Convergence when dwnd non-zero. Compound TCP, 200 Mbps link, 200 ms RTT,  $0.1 \times \text{BDP}$  buffer (Vista, Dummynet).

the buffer is large enough that dwnd decreases to zero when the buffer fills.

When the buffer is smaller, the results are significantly different. It appears that Compound TCP has slightly better RTT-fairness than Reno, while Illinois fares slightly worse, although no confidence intervals have been calculated.

These results have been taken at the comparatively low rate of 100 Mbit/s, to allow experiments of only half an hour. (As discussed above, the congestion epoch duration increases with the bandwidth-delay product, requiring longer experiments for the same accuracy.) Evaluating performance at higher bit rates is left as future work.

## VI. TCP FRIENDLINESS

New congestion control algorithms are required to coexist with legacy TCP flows. In this section we briefly investigate the ‘‘TCP friendliness’’ of TCP Illinois and Compound TCP. Due to lack of space presentation of more comprehensive measurements is left as future work.

Figure 8 illustrates the behaviour of a TCP Illinois flow and a Reno flow sharing a 100 Mbps link. Figure 9 shows the corresponding results for Compound TCP. In these tests both TCP Illinois and Compound TCP are approximately fair with Reno. A notable exception is Figure 8(a) where it can be seen that TCP Illinois is significantly less friendly to the legacy TCP flow than is Compound TCP. This appears to be due to an implementation issue with TCP Illinois whereby the maximum RTT is over-estimated. Since the backoff factor of TCP Illinois is adjusted based on the measured delay relative to maximum RTT, this leads to a reduced backoff and unfairness.

## VII. CROSS TRAFFIC

We also consider the impact of cross-traffic and reverse-path traffic on throughput. With one long-lived flow with RTT 120 ms in the ‘‘forward direction’’ of a 400 Mbit/s link, three cases are studied:

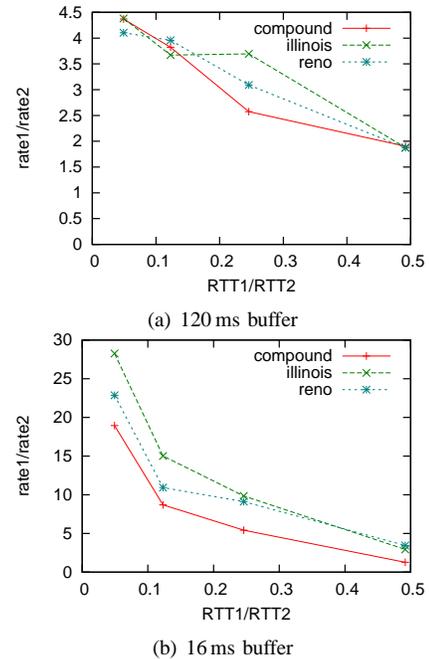
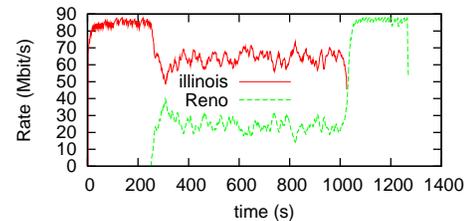
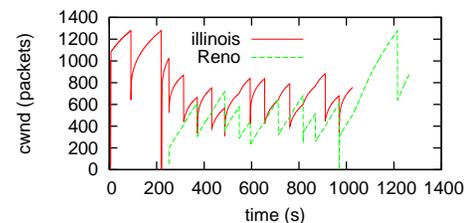


Fig. 7. RTT fairness experiments, 100 Mbit/s link, RTT of flow 2 is 120 ms, RTT of flow 1 is marked on x-axis.



(a) 100 Mbps, 30 ms,  $0.5 \times \text{BDP}$  queue, 65 Mbit/s Illinois, 24 Mbit/s Reno



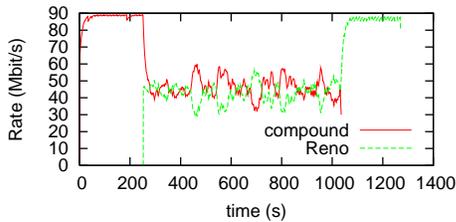
(b) 100 Mbps, 100 ms,  $0.6 \times \text{BDP}$  queue, 53 Mbit/s Illinois, 37 Mbit/s Reno

Fig. 8. Illinois TCP friendliness vs BDP. Upper plot shows averaged cwnd and rates. Lower plot shows raw cwnd history since in this larger BDP example the variations in cwnd are slow enough to be give additional information. (WAN-in-lab)

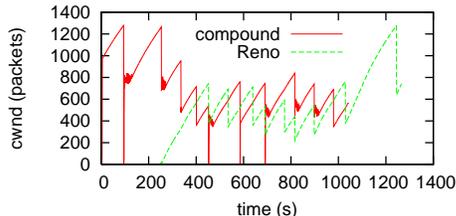
- 1) no cross traffic
- 2) 10% forward TCP cross traffic
- 3) 10% forward TCP cross traffic, long-lived reverse flow.

The TCP cross traffic is of the sort used in Section V, with flow RTTs evenly divided between 30, 60, 90 and 120 ms. The cross traffic uses the Linux Reno algorithm without SACK. For comparison, results for a loss-based high-speed algorithm are also presented. Cubic [2] was chosen since it is the current default in Linux.

Figure 10 shows the mean throughput of the forward flow in each case, taken between 200 s and 1800 s. Note that the total volume of cross traffic is equal in each case (40 Mbit/s),



(a) 100 Mbps, 30 ms,  $0.5 \times \text{BDP}$  queue, 47 Mbit/s Compound, 42 Mbit/s Reno



(b) 100 Mbps, 100 ms,  $0.6 \times \text{BDP}$  queue, 48 Mbit/s Compound, 41 Mbit/s Reno

Fig. 9. Compound TCP friendliness vs BDP. (WAN-in-lab)

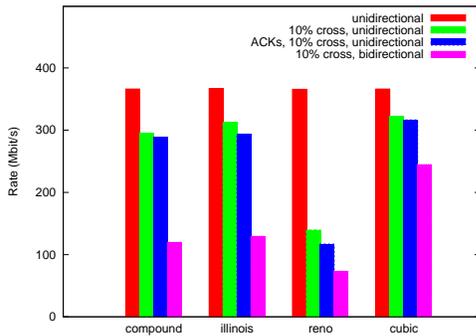


Fig. 10. Throughput of long-lived forward flow for a range of cross traffic scenarios. 400Mbps link, 4096 packet buffer ( $1 \times \text{BDP}$  for 120ms RTT).

and so these figures directly indicate link utilization. When cross traffic is present (green bar), the hybrid algorithms' throughput is reduced by slightly more than the 40 Mbit/s cross traffic volume, and slightly more than the loss-based high-speed algorithm.

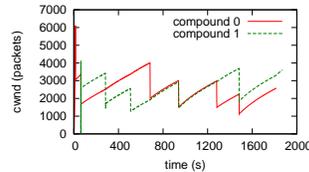
Reverse traffic causes a significant reduction in throughput (magenta bar). Possible reasons include (i) queueing on the reverse path, as investigated in Figure 2, (ii) the small ACKs from the reverse flow reduce the effective size of the buffer, since it holds a fixed number of packets, (iii) burstiness caused by ACK compression. An additional test (blue bar) was performed with 52 byte packets in the forward direction at the rate of reverse-flow ACKs (5 Mbit/s). The similarity between this throughput and that of case 2 shows that the ACKs are probably not the source of the reduction. Since the small reduction incurred by the loss-based high-speed algorithm reflects the impact of ACK compression, presumably the remaining reduction incurred by the hybrid algorithms is due to reverse path queueing.

It is also important to understand the impact that the long-lived flow has on the cross traffic. Table I shows the mean throughput achieved by the cross-traffic in case 2 above (green bar). The top row shows the mean rate of the individual flows, which is weighted towards the many short cross-traffic flows,

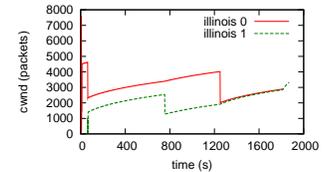
Algorithm	Compound	Illinois	Reno	Cubic
$E[b/T]$ (Mbit/s)	0.698	0.520	0.771	0.310
$E[b]/E[T]$ (Mbit/s)	1.17	0.807	1.32	0.502

TABLE I

RATES OF CROSS TRAFFIC SHARING WITH A LONG-LIVED FLOW OF 120 MS RTT. HERE,  $b$  IS THE NUMBER OF BITS IN A FLOW, AND  $T$  IS THE FLOW DURATION. BOTTLENECK: 400 MBIT/S. TOTAL CROSS TRAFFIC: 10%



(a) Compound TCP



(b) TCP Illinois

Fig. 11. Sensitivity to errors in baseRTT estimate. 150 Mbit/s, 100 ms,  $1.6 \times \text{BDP}$  buffer. (Linux, WAN-in-Lab)

while the bottom row shows ratio of the mean size to mean duration, which is weighted towards the longer flows. Compound TCP gives the highest rate to cross traffic. In contrast, Illinois is consistently more aggressive. Both give greater throughput than the loss-based cubic algorithm, although from Figure 2 it can be seen that the long flow obtains similar throughput in each case. Reno gives cross traffic the highest throughput, since the long-lived flow obtains very little throughput.

The Vegas delay-based algorithm is known to be sensitive to errors in estimating the minimum RTT ("baseRTT") when existing flows cause persistent queueing [14]. It is therefore interesting to investigate the sensitivity of the hybrid TCP Illinois and Compound TCP to errors in the estimate of baseRTT. Fig. 11 shows a typical example from our tests. It can be seen that Compound TCP and TCP Illinois achieve reasonable fairness despite over-estimating baseRTT by 20 ms. The insensitivity of fairness to errors in baseRTT appears to arise because both algorithms are able to detect that queueing is present, despite misjudging the exact magnitude of the queueing, and revert to conservative operation.

## VIII. IMPLEMENTATION ISSUES

### A. TCP Illinois in Linux 2.6.23

Initial tests highlighted a bug leading to incorrect backoff in TCP Illinois, which has now been fixed. We also observed that TCP Illinois commonly over-estimates the maximum RTT. Errors in max RTT estimation appear to arise because Linux timestamps packets at the TCP layer, rather than when they arrive at the NIC. Processing delays were observed to introduce significant delays, e.g., sender side processing of the list of unacknowledged packets was found to introduce significant delays in machines with a 0.5 MB cache, although less so in machines with a 2 MB cache. Outliers in measured RTT cause TCP Illinois to act as if the buffer is very large and thus to treat all regular RTT samples as if queueing is negligible. Hence TCP Illinois becomes overly aggressive, see for example Figure 12.

To mitigate these effects, we patched TCP Illinois to ignore RTT samples for two RTTs after loss. Note, however, that the maximum RTT seems intrinsically a much more fragile

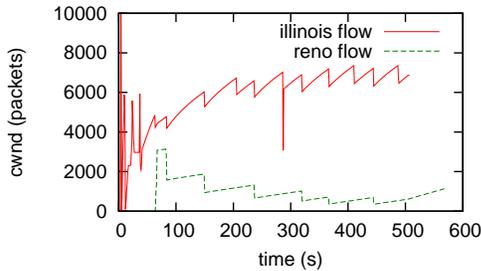


Fig. 12. TCP Illinois with an over-estimate of the maximum RTT, being very unfair to Reno. 400 Mbit/s, 120 ms, 1 BDP.

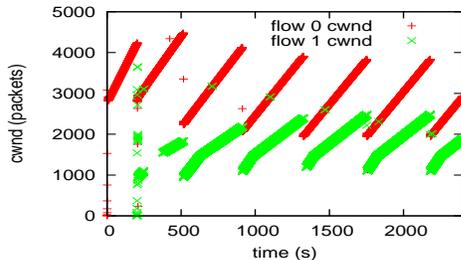


Fig. 13. Anomalous behaviour in Vista SP1 implementation of Compound TCP. 250 Mbit/s link, 100 ms,  $2 \times$ BDP buffer.

statistic than the minimum RTT, since the former is lower bounded by the propagation delay. The maximum RTT may also be susceptible to spoofing by deliberately delaying ACKs, which would allow a receiver to obtain an unfairly high rate.

### B. Compound TCP in Microsoft Vista SP1

A number of issues were highlighted during our tests.

1) In versions prior to Service Pack 1 (SP1), Vista has a serious bug in both the Reno and Compound TCP implementations that leads to cwnd backing off to one packet at every loss event. This bug appears to be fixed in SP1.

2) In Vista SP1 there remain some outstanding issues. For example Fig. 13 shows measurements with two Compound TCP flows sharing a link with a relatively large buffer ( $2 \times$ BDP). For the second flow, the additive increase rate of the loss-based component is consistently less than that of the first flow, leading to persistent unfairness. Observe also the change in the slope of cwnd around times 500 s, 1000 s for the second flow, which does not appear to be correct Compound TCP behaviour.

3) We compared measurements for the Vista implementation of Compound TCP and for a Linux re-implementation [10] based on the Compound TCP internet draft document [11]. This highlighted a number of ambiguities within the internet draft (e.g., the way in RTT is measured and filtered is not specified) in addition to the issues noted above.

4) We observed a significant deterioration in performance at link speeds above around 400Mbps, including spurious backoffs (without packet loss) and backoffs by a factor less than 0.5. We suspect that, similarly to Linux, this is associated with the end host processing burden at higher speeds.

### C. Very high speeds

In preliminary tests at 1 Gbit/s, all of the algorithms tested exhibit irregular behaviour, presumably due to spikes in CPU

activity. End users wishing to choose which implementation to run may be interested to know that on a 122ms 1 Gbit/s Packet-over-Sonet path with 5% bidirectional Pareto cross traffic and a 1 BDP buffer, Linux Compound TCP achieved 809 Mbit/s and TCP Illinois achieved 783 Mbit/s.

## IX. CONCLUSIONS

This paper presents initial experimental results for TCP Illinois and Compound TCP. These tests are for relatively simple scenarios yet they are sufficient to highlight some interesting issues. Both TCP Illinois and Compound TCP can exhibit poor scaling behaviour as path BDP increases. This relates both to their response functions (e.g. even light reverse path traffic makes Compound TCP revert to a Reno-like scaling behaviour, while the linear increase used in TCP Illinois limits scalability) and congestion epoch duration: for Compound TCP this is dominated by the Reno-like loss-based component, for TCP Illinois the concave cwnd evolution leads to poor scaling of congestion epoch duration when buffers are sized proportional to BDP. As a result link utilisation can be low and network responsiveness can become sluggish as BDP increases. We also document a number of important implementation issues observed during our tests.

## X. ACKNOWLEDGEMENTS

This work was supported by Cisco Systems, by Science Foundation Ireland grants 07/IN.1/1901 and 04/IN3/1460, and by National Science Foundation grant 0303620.

## REFERENCES

- [1] R. N. Shorten and D. J. Leith. H-TCP: TCP for high-speed and long-distance networks. in *Proc. PFLDnet*, Argonne, 2004.
- [2] I. Rhee, L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. In *Proc. PFLDnet*, 2005.
- [3] D. Wei, C. Jin, S. Low and S. Hegde FAST TCP: Motivation, architecture, algorithms, performance. *IEEE Trans. Network.*, 14(6):1246–1259, 2006.
- [4] K. Tan, J. Song, Q. Zhang, M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *Proc. INFOCOM*, 2006.
- [5] S. Liu, T. Başar, R. Srikant. TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks.
- [6] F. Vacirca, A. Baiocchi and A. Castellani. YeAH-TCP: Yet Another Highspeed TCP. In *Proc. PFLDnet2007*, 2007.
- [7] H. Shimonish, T. Hama and T. Murase. TCP-Adaptive Reno for Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm, In *Proc. PFLDnet*, Feb. 2006, pp. 87–91.
- [8] S. Floyd. HighSpeed TCP for large congestion windows. RFC 3649, December 2003.
- [9] G. S. Lee, L. L. H. Andrew, A. Tang and S. H. Low. A WAN-in-Lab for protocol development. in *Proc. PFLDnet*, Feb. 2007, pp. 85–90.
- [10] [http://netlab.caltech.edu/lachlan/ctcp/ctcp\\_2\\_6\\_23.patch](http://netlab.caltech.edu/lachlan/ctcp/ctcp_2_6_23.patch)
- [11] M. Sridharan, K. Tan, D. Bansal, D. Thaler. Internet Draft draft-sridharan-tcpm-ctcp-01.txt
- [12] R. N. Shorten, F. Wirth and D. J. Leith A positive systems model of TCP-like congestion control: Asymptotic results *IEEE/ACM Trans. Networks*, 14(3):616–629, June. 2006.
- [13] L. Brakmo and L. Peterson. TCP Vegas: end-to-end congestion avoidance on a global Internet. *IEEE J. Select.Areas Commun.*, 13(8):1465–80, Oct. 1995.
- [14] S. H. Low, L. L. Peterson and L. Wang. Understanding Vegas: A Duality Model. *J. ACM*, 49(2):207–235, Mar. 2002.