Experimental evaluation of Cubic-TCP

D.J. Leith, R.N.Shorten, G.McCullagh
Hamilton Institute, Ireland

***Abstract*— In this paper we present an initial experimental evaluation of the recently proposed Cubic-TCP algorithm. Results are presented using a suite of benchmark tests that have been recently proposed in the literature [12], and a number of issues are of practical concern highlighted.**

I. Introduction

In this paper we present the results of experimental tests on the recently proposed high-speed TCP variant, Cubic-TCP[11]. Consideration of the Cubic algorithm is particularly topical in view of recent discussions regarding its adoption in Linux[8].

In summary we find that,

1. Networks in which Cubic TCP is deployed suffer from slow convergence. The dependence of the cubic increase function on flow $cwnd$ means that flows with higher congestion windows are more aggressive initially than flows with lower congestion windows. The resulting slow convergence behaviour yields poor network responsiveness and prolonged unfairness between flows.
2. In common with other high-speed protocols, Cubic TCP uses an aggressive additive increase action to maintain short congestion epochs on high bandwidth-delay product paths. We find that in unsynchronised environments, the associated cost of "missing a drop" (whereby flows that are not informed of congestion rapidly increase their $cwnd$) is similar for both Cubic TCP and HTCP.
3. At bandwidth-delay products above about 5000 packets, Cubic TCP reverts to a linear increase function. This implies that in high-speed networks the congestion epoch duration eventually scales linearly with BDP (and so similarly to standard TCP).
4. At higher speeds, for buffer sizes below 30% BDP the link utilisation achieved by Cubic TCP collapses to around 50% of link capacity and is significantly lower than the link utilisation achieved by standard TCP. Although it requires further investigation, this behaviour appears to be associated with the generation of large packet bursts by the Cubic TCP algorithm.
5. For flows with different RTTs, Cubic exhibits unfairness that is strongly dependent on the start time of the flows. It is unclear at present why this non-convergence behaviour occurs – it may be due to a fundamental stability issue or perhaps associated with implementation issues.

II. Experimental Setup

A. Hardware and Software

All tests were conducted on an experimental testbed. Commodity high-end PCs were connected to gigabit switches to form the branches of a dumbbell topology, see Figure 1.

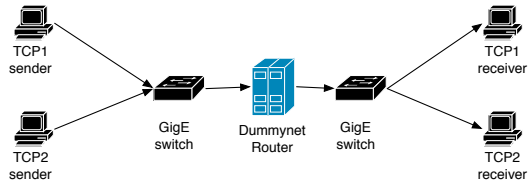

Fig. 1. Experimental set-up.

	Description
CPU	Intel Xeon CPU 3.00GHz 1066 FSB
Memory	1 Gbyte
Motherboard	Dell PowerEdge PE860
Kernel	Linux 2.6.18 with Cubic bug fix[9]
txqueuelen	1000
max_backlog	2500
NIC	Intel Pro 1000PT PCIe x4
NIC Driver	e1000 5.2.52-k4
TX & RX Descriptors	4096

TABLE I
Hardware and Software Configuration.

Although all are not shown in Figure 1, a total of 14 end hosts are available for traffic generation. All sender and receiver machines used in the tests have identical hardware and software configurations as shown in Table I and are connected to the switches at 1Gb/sec. The router, running the FreeBSD dummynet software, can be configured with various bottleneck queue-sizes, capacities and round trip propagation delays to emulate a wide range network conditions.

Apart from the router, all machines run an instrumented version of the Linux 2.6.18 kernel. We note that the implementation of Cubic-TCP included in the Linux 2.6.18 kernel distribution and earlier is known [9] to be incorrect (this has subsequently been corrected). In our tests we use a corrected implementation. To provide consistency, and control against the influence of differences in implementation as opposed to differences in the congestion control algorithm itself, we use a common kernel for all tests. It is known that the at high bandwidth-delay products SACK processing etc in the Linux network stack can impose a sufficiently high burden on end hosts that it leads to a significant performance degradation [1], [12]. We performed tests to confirm, on our hardware, appropriate network stack operation over the range of network conditions tested.

The kernel is instrumented using custom RelayFS monitoring to allow measurement of TCP variables.

In order to minimise the effects of local hosts queues and flow interactions, unless otherwise stated we only ran one

long-lived flow per PC with flows injected into the testbed using `iperf`. Web traffic sessions are generated by dedicated client and server PCs, with exponentially distributed intervals between requests and Pareto distributed page sizes. This is implemented using a client side script and custom CGI script running on an Apache server. Following [10], unless otherwise stated, we used a mean time between requests of 1 second and a Pareto shape parameter of 1.2 and mean 6.0. Each individual test was run at least ten minutes each. Tests collecting statistics on unsynchronised operation were run for at least one hour in order to ensure reliable statistics. In the case of tests involving Standard TCP, we ran individual tests for up to an hour as the congestion epoch duration becomes very long on large bandwidth-delay products paths.

B. Comparative Testing

Our test setup corresponds to that in [12] and hence the Cubic TCP measurements reported here can be directly compared with previous measurements reported for Standard TCP, High-Speed TCP, Scalable TCP, BIC-TCP, FAST-TCP and H-TCP.

C. Range of Network Conditions

Similarly to [12], in this paper we consider round-trip propagation delays in the range 16ms-200ms and bandwidths ranging from 1Mb/s-500Mb/s. We do not consider these values to be definitive – the upper value of bandwidth considered can, in particular, be expected to be subject to upwards pressure. We do, however, argue that these values are sufficient to capture an interesting range of network conditions that characterises current communication networks, and the behaviour of protocols across this range. In all of our tests we consider delay values of 16ms, 40ms, 80ms, 160ms, 200ms and bandwidths of 1Mb/s, 10Mb/s, 250Mb/s and 500Mb/s. In addition, we perform each test with various levels of competing bidirectional web sessions. This defines a three-dimensional grid of measurement points where, for each value of delay and level of web traffic, performance is measured for each of the values of bandwidth. Owing to space restrictions, we cannot include the results of all our tests here. We therefore present results for a subset of network conditions that are representative of the full test results obtained. A more complete collection of test results will be posted on the Hamilton Institute website.

III. CUBIC TCP ALGORITHM IN LINUX

Before proceeding, we briefly describe the Cubic TCP algorithm used in Linux. Cubic-TCP combines the basic ideas first proposed in High-Speed TCP, and H-TCP. Namely, the $cwnd$ additive increase rate is a function of time since the last notification of congestion (as in H-TCP), and of the window size at the last notification of congestion (similarly to HS-TCP). Pseudo code for the main functionality of the Cubic algorithm is shown in Algorithm 1. The features of this algorithm can be summarised as follows,

1) *Modified slow start*. A modified slow start behaviour is employed at startup. Once $cwnd$ rises above ssthresh (which is initalised to a value of 100 packets in Cubic), Cubic exits normal slow start and changes to use a less aggressive exponential increase where $cwnd$ is increased by one packet for every 50 acks received or, equivalently, $cwnd$ doubles approximately every 35 round-trip times. See lines 25-26 in Algorithm 1.
2) *Backoff factor 0.8*. On packet loss, $cwnd$ is decreased by a factor of 0.8 (compared with a factor of 0.5 in the standard TCP algorithm). See line 44 in Algorithm 1.
3) *Clamp on maximum increase rate*. The additive increase rate during AIMD operation is limited to be at most $20*delay\_min$ packets per RTT, where $delay\_min$ is an estimate of the round-trip propagation delay of the flow. See line 23 in Algorithm 1. Converting from packets per RTT to packets per second, this clamp is roughly equivalent to a cap on the increase rate of 20 packets/s independent of RTT.
4) *Cubic increase function*. Subject to this clamp, the additive increase rate used is $target - cwnd$ packets per RTT. Note that the effect of this increase is to adjust

Algorithm 1 : Pseudo code of main functionality in Linux 2.6.18 Cubic algorithm

```
1:  Initialise:
2:  last_max = 0; loss_cwnd = 0; epoch_start = 0; ssthresh = 100
3:  b = 2.5; c = 0.4
4:
5:  On each ACK:
6:  delay_min = min(RTT, delay_min)
7:  if cwnd < ssthresh then
8:      cwnd++ //slow start
9:  else
10:     if epoch_start = 0 then
11:         epoch_start = current time
12:         K = max(0, ∛(b * (last_max − cwnd)))
13:         origin_point = max(cwnd, last_max)
14:     end if
15:     t =current time +delay_min − epoch_start
16:     target = origin_point + c * (t − K)^3
17:     if target > cwnd then
18:         cnt = cwnd/(target − cwnd)
19:     else
20:         cnt = 100 * cwnd
21:     end if
22:     if delay_min > 0 then
23:         cnt = max(cnt, 8 * cwnd/(20 * delay_min)) //max AI rate
24:     end if
25:     if loss_cwnd == 0 then
26:         cnt=50 // continue exponential increase before first backoff
27:     end if
28:     if cwnd_cnt > cnt then
29:         cwnd++
30:         cwnd_cnt = 0
31:     else
32:         cwnd_cnt++
33:     end if
34: end if
35:
36: On packet loss:
37: epoch_start = 0
38: if cwnd < last_max then
39:     last_max = 0.9 * cwnd
40: else
41:     last_max = cwnd
42: end if
43: loss_cwnd = cwnd
44: cwnd = 0.8 * cwnd // backoff cwnd by 0.8
```

$cwnd$ to be equal to $target$ over the course of a single RTT. The value of $target$ is calculated (see line 16 in algorithm) from:

$$target = W_{max} + c(t - \sqrt[3]{(b(W_{max} - 0.8W))^3} \quad (1)$$

where t is the elapsed time since last backoff (approximately – the value of $delay\_min$ is added to this value, see line 15) and W_{max} is related to the cwnd at last backoff and is denoted $origin\_point$ in the code. W is the $cwnd$ value immediately before the last backoff, so that $0.8W$ is the $cwnd$ value just after backoff has occurred.

5) *Adaptation of cubic function.* The value of W_{max} is adjusted depending on whether the last backoff occurred before or after $cwnd$ reached the previous W_{max} value. Let W denote the $cwnd$ value immediately before backoff. Then, W_{max} is set equal to the W when W is larger than the previous value of W_{max}. Otherwise W_{max} is set equal to $0.9W$. See lines 38-42 in algorithm.

The Linux Cubic algorithm also includes code which ensures that the Cubic algorithm is at least as aggressive as standard TCP, but this plays little role in our discussion and we refer the interested reader to the Linux code for further details.

A. Other Cubic Variants

It is important to note that a number of variants of the Cubic TCP algorithm exist. In this paper we focus on the algorithm contained in the standard Linux distribution as this is both the most recent variant and the variant in production use. This Linux Cubic algorithm differs from that described in the original Cubic paper [11], and from algorithms used and documented in recent tests. Amongst other things, the original cubic algorithm proposed in [11] lacks the clamp on increase rate and the mode switch based on the parameter W_{max} in Algorithm 1. The experimental results reported by the Cubic authors[3] make use of a custom patch to the Linux 2.6.13 kernel[1] which differs from Algorithm 1.

IV. Fairness with same RTT

We begin by considering the simplest case of flows sharing a single bottleneck and with the same round-trip time. A basic requirement is for flows to be allocated bandwidth in a fair manner in this situation. Figure 2 shows typical measured time histories with Cubic TCP following startup of a second flow. A number of features are immediately evident.

- *Slow convergence, prolonged unfairness.* It can seen from Figure 2 that the $cwnd$s converge very slowly. The rate of convergence decreases as the bandwidth-delay product (BDP) is increased, with convergence times in excess of 300s commonly observed in our tests.
- *Long-term fairness.* Provided tests are run for a sufficiently long period, our measurements indicate that the flows do asymptotically converge to an approximately fair share of the bottleneck bandwith. See also Figure 3.
- *Linear increase at high BDPs.* For $cwnd$'s above about 5000 packets it can be seen that the cubic increase function becomes replaced by a simple linear AIMD increase. This is particularly evident for flow 1 in the lower plot in Figure 2. This implies that in high-speed networks the congestion epoch duration eventually scales linearly with BDP. This is similar to standard TCP.

We note that although in the examples shown in Figure 2 the flows are nearly synchronised (i.e. both flows experience drops at most congestion events), we find that the slow convergence behaviour is *not* confined to such situations and that the same qualitative behaviour is evident in unsynchronised environments.

For example, Figure 4 shows the corresponding results when tests are carried out with 200 bi-directional web flows sharing the bottleneck link. Figure 4 plots the throughput time histories of the two long-lived flows, averaged over 25 runs. This is an ensemble average i.e. each time point is the average over 25 runs. We observe that while de-synchronisation of drops sometimes speeds up convergence (e.g. when the newly started flow misses drops), it also sometimes slows convergence (when the incumbent slow misses drops) and on balance this yields similarly slow convergence to synchronised situations.

A. Source of slow convergence.

We can gain some insight into the slow convergence exhibited by Cubic TCP by noting that the cubic increase function depends on the flow $cwnd$ at the last congestion event. Generally, flows with larger $cwnd$'s are more aggressive than flows with smaller $cwnd$'s, and are consequently more able to acquire bandwidth as it becomes available. New flows are thus at a disadvantage and sustained unfairness can occur. We comment that a similar feature has previously been highlighted in other high-speed protocols, which are also known to exhibit slow convergence [5], [4], [6], [7].

The key here is that the convergence behaviour is determined by *interactions* between competing flows. Hence, "bath of noise" type mean-field analysis such as that used in the popular Padhye fluid model provides little insight. Roughly speaking, the convergence rate of the network depends on two factors: (a) the rate at which individual flows release bandwidth when informed of congestion; and (b) the rate at which individual flows acquire available bandwidth. The first depends on the backoff factors employed in the network, the second on the additive increase strategy employed by the individual sources. Together they determine the net rate at which bandwidth can be distributed in the network.

In standard TCP flows with the same RTT probe for available bandwidth at the same rate. Strategies such as H-TCP seek to mimic this behaviour by making all sources probe in the same manner (at least in the synchronised case). Other strategies, such as High-Speed TCP, BIC-TCP and Cubic-TCP, result in flows with larger window sizes probing more aggressively than those with smaller windows. This basic asymmetry between flows increase rates has the effect of slowing convergence.

[1]See `http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/tiny_release/installbic_2.6.13.htm`

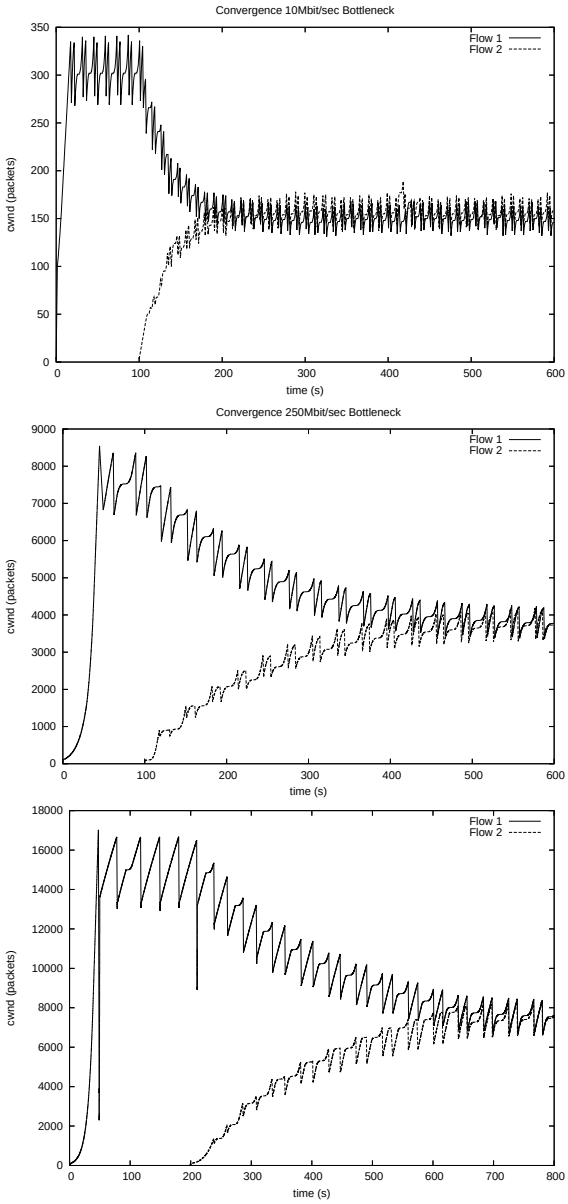

Fig. 2. Cubic TCP cwnd time histories following startup of a second flow. Bandwidth is 10Mbits/s (top), 250 Mbit/sec (middle) and 500 Mbit/sec (bottom). RTT is 200ms, queue size 100% BDP, no web traffic.

This effect is reinforced by changes to the AIMD backoff factor. In standard TCP flows backoff $cwnd$ by 0.5 on detecting packet loss. Strategies such as BIC-TCP and Cubic-TCP instead use a backoff factor of 0.8. As a result, flows release bandwidth more slowly when informed of congestion, again having the effect of slowing convergence.

B. Slow convergence implies prolonged unfairness.

One consequence of slow convergence is that periods of extreme unfairness between flows may persist for long periods; even in situations where flows do eventually converge to fairness. Such situations are masked when fairness results are presented purely in terms of long-term averages. However, this behaviour is immediately evident, for example, in the time histories shown in Figure 2 and it seems clear that it has important practical implications. For example, two identical file transfers may have very different completion times depending on the order in which they are started. Also, long-lived flows can gain a substantial throughput advantage at the expense of shorter-lived flows. The latter seems particularly problematic as the majority of TCP flows are short to medium sized and so a single long-lived flow may potentially penalize a large number of users (akin to a form of denial of service).

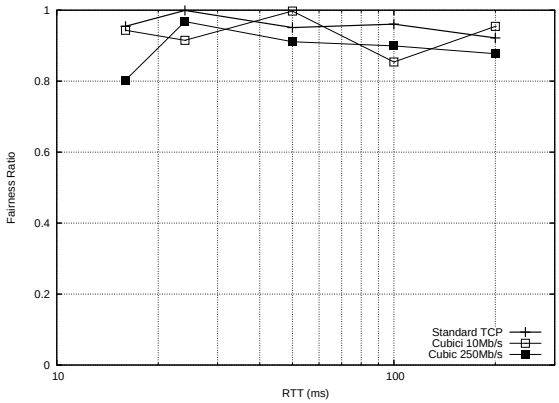

Fig. 3. Ratio of throughputs of two Cubic TCP flows with the same RTT (also sharing same bottleneck link and operating same congestion control algorithm) as path propagation delay is varied. Flow throughputs are averaged over the last 200s of each test run and so approximate asymptotic behaviour, neglecting initial transients. Results are shown for 10Mbit/sec and 250Mbit/sec bottleneck bandwidths. The bottleneck queue size is 100% BDP, no web traffic.

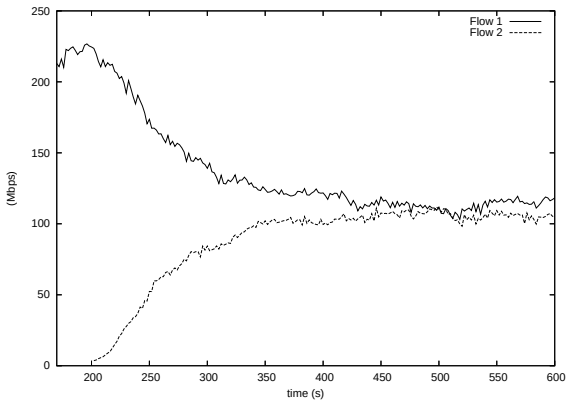

Fig. 4. Impact of web traffic on convergence. Evolution of mean bandwidth, averaged over 20 test runs, following startup of a second flow. 200 background web flows (100 in each direction). Link bandwidth is 250 Mbit/sec, RTT is 200ms, queue size 100% BDP.

With regard to the last point, the impact of a long-lived flow on a short-lived flow is illustrated, for example, in Figure 5. Here, we measure the completion time for a download versus the size of the download. Measurements are shown (i) for the baseline case where no other flow shares the bottleneck link and (ii) for the case where a single long-lived flow shares the link and competes for bandwidth. It can be seen that in the baseline situation, Cubic-TCP, standard TCP and H-TCP all exhibit similar completion times. It is perhaps initially surprising that standard TCP performs so well in this test, in view of concerns about performance in high-speed paths. However, we note that the link in this example is provisioned with a BDP of buffering. A standard TCP flow slow-starts to

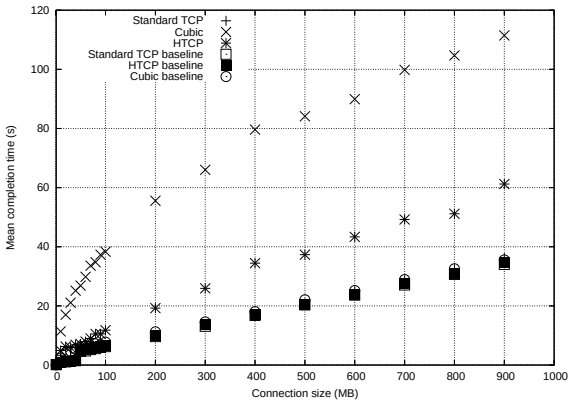

Fig. 5. Completion time vs connection size for (i) baseline case with no competing flows and (ii) with a competing long-lived flow. Measurements are shown for standard TCP, Cubic-TCP and H-TCP. Bandwidth is 250 Mbit/sec, RTT is 50ms, queue size 100% BDP.

fill the pipe and when it then backs off $cwnd$ the buffer just empties and the link remains fully utilised, hence achieving low completion times. More interesting is the performance when a long-lived flow is present. It can be seen that the completion time of the short-lived flow approximately doubles with H-TCP, as might be expected with two flows now sharing the link. In contrast, the completion time for the short-lived flow increases by more than a factor of four. This arises because of the sluggishness with which the long-lived flow releases bandwidth to the short-lived flow, so that the short-lived flow is starved of bandwidth for a prolonged period. It is interesting to also contrast this behaviour with that of standard TCP. It can be seen that the completion time of the short-lived flow is essentially the same both in the baseline and competing long-lived flow cases. Further inspection reveals that the throughput of the long-lived flow is much lower (by a factor 5) compared with that of the long-lived flow when using H-TCP. The long-lived standard TCP flow releases bandwidth but is very slow to regain throughput following losses induced by startup of a short-lived flow.

The impact of slow convergence is not confined to situations where new flows start up but is of importance more generally. Figure 6 plots example $cwnd$ time histories measured in conditions where packet drops are unsynchronised. While the long-term (measured over a period of an hour) throughputs of the flows are similar, it can be seen that sustained periods (extending to hundreds of seconds) of unfairness occur. What is happening here is that when a flow misses a drop, it is able to grab a larger share of the available bandwidth and, owing to the slow convergence behaviour of the congestion control algorithm, the resulting unfairness is able to persist for long periods.

C. Impact of "missing a drop"

A common feature of loss-based high-speed protocols is their aggressive additive increase phase. This is perfectly natural as it is the primary mechanism for ensuring that the time between congestion events remains small on high BDP paths. However, a consequence of this action is that flows are able to rapidly grab additional bandwidth as a result of

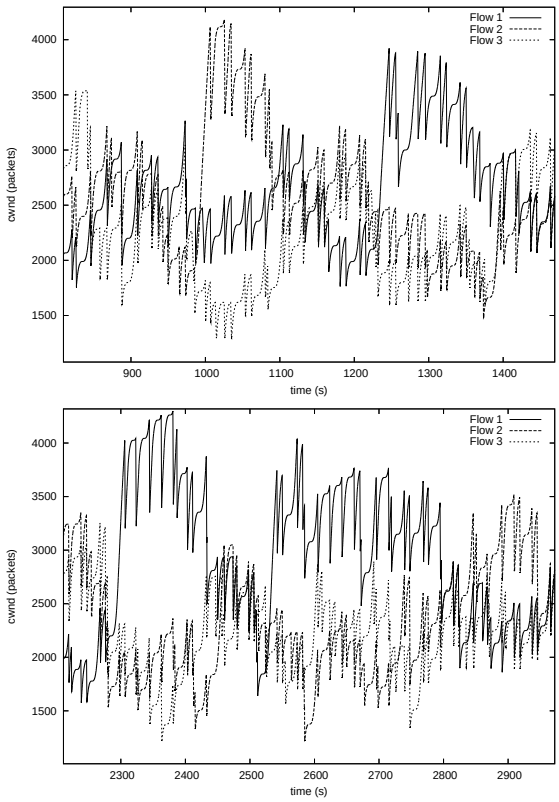

Fig. 6. Two examples of Cubic TCP cwnd time histories. Three long-lived flows sharing a bottleneck link with 25 on-off sessions (mean time between requests of 10 seconds, connection sizes are Pareto distributed with shape parameter of 1.2 and mean 600 packets). Bandwidth is 250 Mbit/sec, RTT is 200ms, queue size 100% BDP.

missing a drop at a network congestion event. This issue has been previously discussed by a number of authors, e.g. see [2] and references therein.

We begin here by first noting that under synchronised conditions, where every flow sees packet loss at every congestion event, the issue of missing a drop does not arise. The increase function used by the original Cubic TCP algorithm[11] is tailored to synchronised conditions. Namely, the inflection point (centre point of the flat section) of the cubic increase curve is placed at the $cwnd$ value at which the last backoff occurred. Under synchronised conditions this leads to concave shape. The Cubic TCP algorithm algorithm as implemented in Linux is modified to adapt the inflection point based on whether the $cwnd$ at backoff is increasing or decreasing compared to its value at the last backoff, see lines 38-42 in Algorithm 1. This adaptation improves the convergence rate of Cubic TCP over the original algorithm. Under synchronised conditions this adaptive action leads to a period two cycle, illustrated in Figure 7 and is also evident in many of the $cwnd$ time histories shown elsewhere in the paper. Observe that in every second period the cubic increase is aggressively probing for bandwidth at the point of backoff.

Consider now the situation where flows are not synchronised. When a Cubic TCP flow misses a drop, the cubic increase function continues past the inflection point and probes for additional bandwidth at a much faster rate than flows recently experiencing a drop. This probing action is cubic in

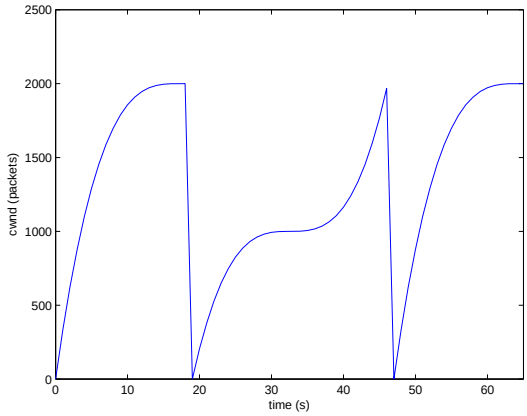

Fig. 7. Example of Cubic TCP $cwnd$ evolution under synchronised conditions.

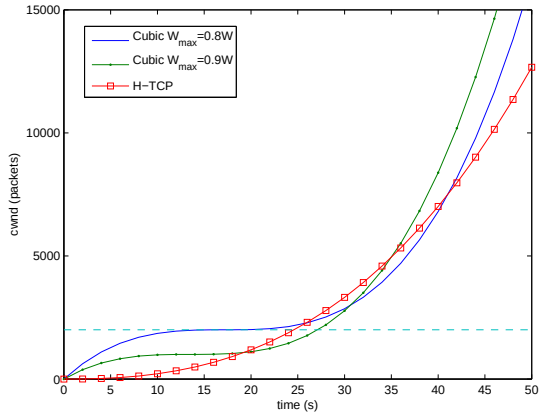

Fig. 8. Cubic TCP increase function for the situation where the $cwnd$ at last backoff is 10000 packets. The y-axis is normalised so that the origin lies at the $cwnd$ immediately after backoff, the dashed line then marks the normalised $cwnd$ at last backoff. It can be seen that the inflection point of the *Cubic* $W_{max} = 0.8W$ curve is located at this value.

shape i.e. $cwnd$ increases as a cubic function. An example of the resulting cubic increase is shown in Figure 8. The curves marked *Cubic* $W_{max} = 0.8W$ and *Cubic* $W_{max} = 0.8W$ in Figure 8 correspond, respectively, to the two possible increase functions that can occur due to the adaptation of the inflection point noted previously. Note that due to the period two nature of cubic $cwnd$ evolution, missing a drop on the second period leads immediately to an aggressive cubic increase as the inflection point occurs earlier. In both cases the potential for large excursions is evident, and is demonstrated experimentally in Figure 10.

For comparison, also plotted on Figure 8 is the $cwnd$ increase function used by H-TCP. This is also a cubic increase function, although lacking an inflection point. It can be seen that on the right-hand side of the plot, corresponding to the behaviour after missing a drop, the H-TCP cubic function is similar to that of Cubic TCP[2].

To confirm the similar natures of the Cubic-TCP and H-TCP increases functions in unsynchronised environments, Figure 9 plots the measured $cwnd$ distributions for both algorithms. To control for the differences in backoff factor used in Cubic and the standard H-TCP algorithm, measurements are taken using a backoff factor of 0.8 for H-TCP (but without any other change to the algorithm). It can be seen that the $cwnd$ distributions are extremely similar, as might be expected from the foregoing discussion. Note that this is in line with previous simulation results reported in [2]. It appears that reported differences between the coefficient of variation of the $cwnd$ distributions of Cubic-TCP and H-TCP may therefore be primarily[3] attributed to the differences in backoff factors used by the algorithms, rather than to the increase functions.

On a broader note, a reasonable question, not yet answered, is whether it in fact matters that high-speed TCP flows may

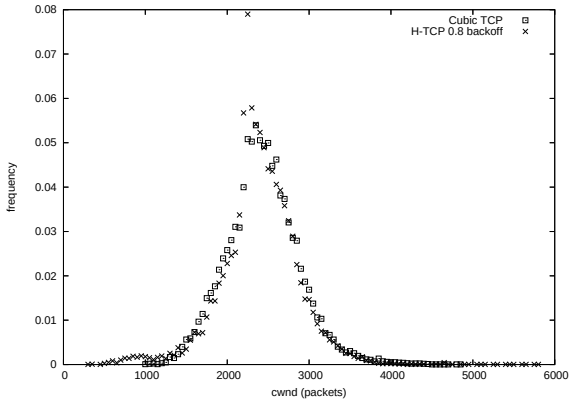

Fig. 9. Measured distribution of $cwnd$. Measurements are shown for both Cubic-TCP and H-TCP using a backoff factor of 0.8. Experimental setup as in Figure 6 – three Cubic flows and 25 background sessions. Bandwidth is 250 Mbit/sec, RTT 200ms, queue size 100% BDP.

exhibit large variations in $cwnd$ under unsynchronised conditions. Firstly, we note that network paths contain extensive buffering and so variations in $cwnd$ need not translate into variations in throughput. Most applications also use receive buffers to further smooth out variations in arriving traffic. Secondly, TCP is designed for best-effort traffic. Traffic that requires a nearly constant arrival rate, and which cannot use buffering to ensure this, should probably use a congestion control algorithm specifically targetted at this requirement, such as TFRC. With this in mind, it seems reasonable to argue that the most important features to consider when designing new TCP congestion control algorithms are (i) the ability to send packets quickly (short file transfer times), (ii) in a fair manner (avoiding prolonged unfairness), and (iii) without causing congestion collapse (maintaining the integrity of the Internet). In this context, issues such as convergence rate, fairness, the ability to fill the network pipe quickly, and how these properties scale with increasing bandwidth, are surely paramount, and fluctuations in flow $cwnd$'s area are a minor secondary issue.

[2]The H-TCP cubic function is slightly less aggressive than that of Cubic TCP. Since $cwnd$ increases slowly on the flat section in the Cubic TCP increase curve, this must be compensated for by a more aggressive increase rate in order to maintain a low congestion epoch duration.

[3]But possibly not solely. For example, different algorithms can yield different patterns of packet drops under the same network conditions, and thereby affect the degree of synchronisation/unsynchronisation – this sort of issue is well known in the context of pacing in standard TCP.

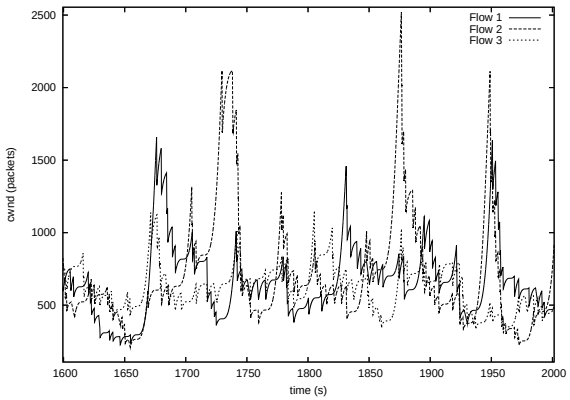

Fig. 10. Example of Cubic TCP cwnd time histories with three long-lived flows sharing a bottleneck link with 50 on-off sessions (mean time between requests of 10 seconds, connection sizes are Pareto distributed with shape parameter of 1.2 and mean 600 packets). Bandwidth is 250 Mbit/sec, RTT is 200ms, queue size 100% BDP.

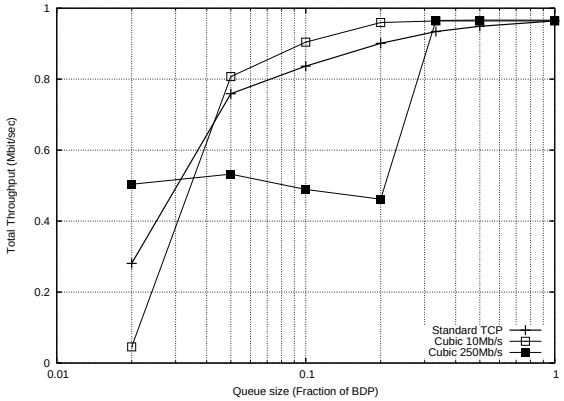

Fig. 11. Aggregate throughput of two competing Cubic TCP flows with 10Mbit/sec and 250 Mbit/sec bottleneck bandwidths. Both flows have end-to-end round-trip propagation delays of 100ms.

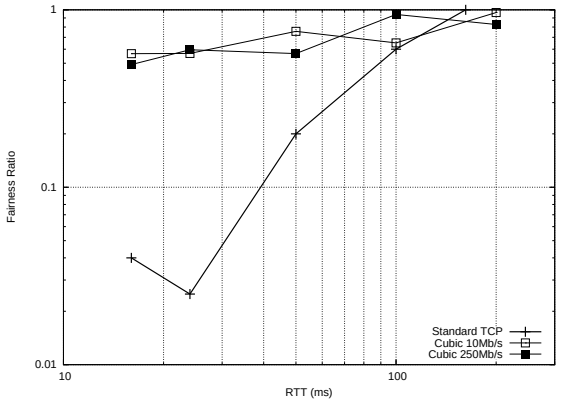

Fig. 12. Ratio of throughputs of two competing Cubic TCP flows as the propagation delay of the second flow is varied. Results are shown for 10Mbit/sec and 250Mbit/sec bottleneck bandwidths. Flow 1 has RTT of 162ms, the RTT of Flow 2 is marked on the x-axis of the plots. Queue size is 100% BDP, no web traffic.

V. Efficiency

To evaluate the link utilisation of Cubic TCP, we consider two flows having the same propagation delay and measure average throughput as the buffer provisioning is varied from 2.5% to 100% of the bandwidth-delay product, see Figure 11. Results are shown for a 10Mb/s and a 250Mb/s link. As a reference, also plotted on Figure 11 is the efficiency for standard TCP.

In the case of a 10Mb/s link, it can be seen that for buffers sized about 5% BDP Cubic TCP achieves greater link utilisation than standard TCP. This is to be expected owing to the larger AIMD backoff factor of 0.8 used by Cubic as opposed to the backoff factor of 0.5 used by standard TCP (so that Cubic decreases $cwnd$ by less than standard TCP on detecting packet loss). At buffer sizes below 2.5%, the link utilisation achieved by both standard TCP and Cubic TCP falls substantially, presumably due to micro-scale packet bursts flooding the queue once it reaches such a small size.

Somewhat surprisingly, we observe quite different behaviour at 250Mb/s. It can be seen from the lower plot in Figure 11 that for buffer sizes below 30% BDP the link utilisation achieved by Cubic TCP falls to around 50% of link capacity and is significantly lower than the link utilisation achieved by standard TCP. Although it requires further investigation, this behaviour appears to be associated with the generation of large packet bursts by the Cubic TCP algorithm and might warrant a modified implementation to mitigate this effect.

VI. Fairness with different RTTs

Figure 12 shows the ratio of measured throughputs when the propagation delay of the first flow is held constant at 162ms and the propagation delay of the second flow is varied. Results are shown both for a bottleneck link bandwidth of 10 Mb/s and 250Mb/s. Results are shown when the queue is sized at 100% BDP since, as discussed above, there appear to be additional issues when smaller sized queues are employed. Also marked on Figure 12, for reference, are the corresponding measurements obtained using standard TCP. It can be seen that Cubic TCP is generally significantly more fair than standard TCP, with the ratio of flow throughputs never falling below about 0.25.

It is perhaps unexpected, however, that there is in fact any unfairness at all between Cubic TCP flows as the Cubic TCP increase function used does not depend on flow RTT. That is, the Cubic TCP increase function is defined as a function time in seconds, in contrast to standard TCP where the increase is specified per RTT. On closer inspection, we find that the degree of unfairness in Cubic TCP is strongly dependent on the start time of the flows. See for example Figure 13. It is unclear at present why this behaviour occurs.

We note that at low $cwnd$'s Cubic TCP is observed exhibit gross unfairness. This is illustrated for example in Figure 14, where it can be seen that one flow is essentially starved of bandwidth. This effect appears to be associated with quantisation of the Cubic TCP increase function and $cwnd$ backoff.

VII. Backward Compatibility

Figure 15 plots the ratio of measured throughputs of two flows with the same propagation delay and a shared bottleneck link. The first flow operates the standard TCP algorithm while the second flow operates Cubic TCP variant. Results are shown both for bottleneck link bandwidths of 10 Mb/s and 250Mb/s.

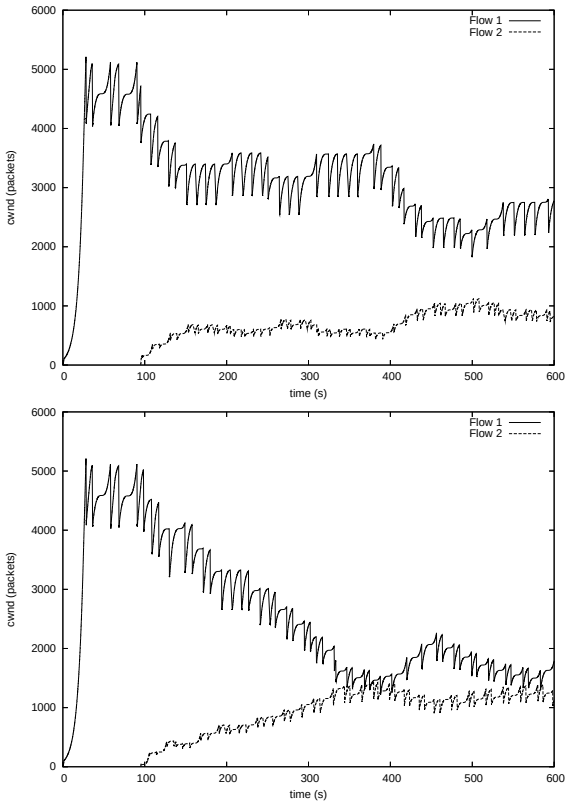

Fig. 13. Cubic TCP $cwnd$ time histories following startup of a second flow. Startup time of second flow is 93s in top plot and 98s in lower plot. 250Mbps link, flow 1 RTT is 16ms, flow 2 RTT is 162ms, 100% BDP queue.

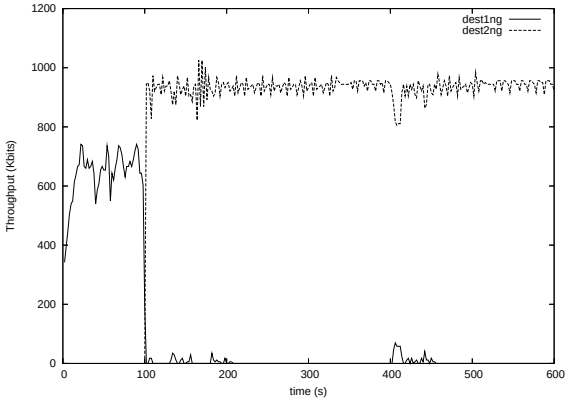

Fig. 14. Two Cubic TCP flows. 1Mbps link, flow 1 RTT 160ms, flow 2 RTT 16ms.

We note that at low $cwnd$'s, Cubic TCP can exhibit gross unfairness when competing with standard TCP flows. See for example, Figure 16. As discussed previously, this appears to be due to quantisation issues in the Cubic TCP algorithm.

VIII. Summary

In this paper we present an initial experimental evaluation of the recently proposed Cubic-TCP algorithm. Results are presented using a suite of benchmark tests that have been recently proposed in the literature [12], and a number of issues are of practical concern highlighted. While some of these issues may be considered minor (including questions surrounding transient $cwnd$ fluctuations), a number are of significant concern. Amongst these, the issue of slow convergence that is a feature of Cubic-TCP, and of other algorithms, seems most worrying.

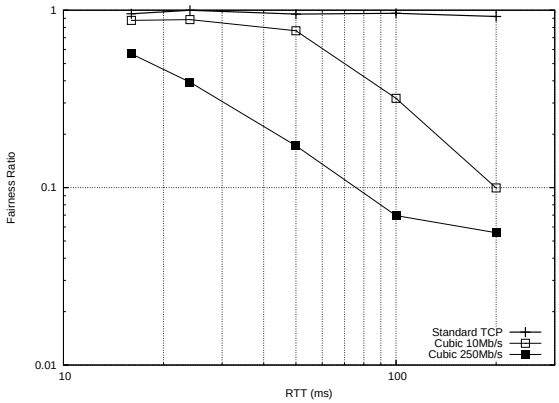

Fig. 15. Ratio of throughputs of competing Cubic TCP and standard TCP flows as path propagation delay is varied. Results are shown for 10Mbit/sec and 250Mbit/sec bottleneck bandwidths. Both flows have the same RTT. Queue size is 100% BDP, no web traffic. Results for two competing standard TCP flows are also shown for reference.

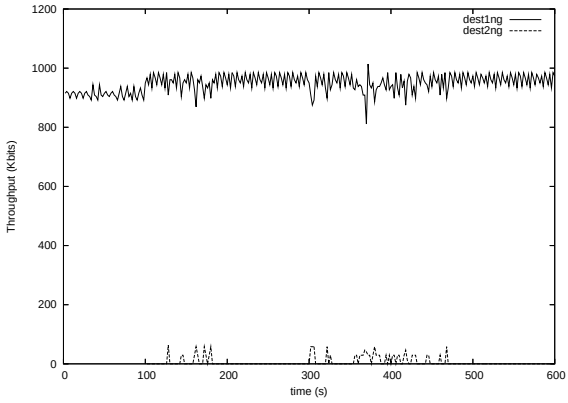

Fig. 16. Cubic TCP (flow 1) and standard TCP (flow 2) flows. 1Mbps link, BDP queue, flow 1 RTT 160ms, flow 2 RTT 16ms.

IX. Acknowledgements

This work was supported by Science Foundation Ireland grants 00/PI.1/C067 and 04/IN3/I460. Discussions and experimental tests carried out by Yee-Ting Lee are gratefully acknowledged.

References

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

[2] D.J.leith and R.N.Shorten. Impact of drop synchronisation on TCP fairness in high bandwidth-delay product networks. In *Proc. Workshop on Protocols for Fast Long Distance Networks, Nara, Japan.*, 2006.

[3] I.Rhee and et al. `http://netsrv.csc.nscu.edu/highspeed/convex-ordering/`.

[4] C. King, R. Shorten, F. Wirth, and M. Akar. Growth conditions for the global stability of highspeed communication networks. Accepted for publication in IEEE Transactions on Automatic Control, 2007.

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

[6] R.N.Shorten, C. King, D.J.Leith, and J.Foy. Modelling TCP in drop-tail and other environments. Accepted for publication in Automatica, 2007.

[7] U. Rothblum and R.N.Shorten. Analysis of high-speed congestion control protocols: A contraction mapping approach. Accepted for publication in SIAM, Control and Optimization, 2007.

[8] S.Hemminger. `http://www.mail-archive.com/netdev@vger.kernel.org/msg23215.html`.

[9] S.Hemminger. `http://kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.18.2`.

[10] W.Willinger, M.S.Taqqu, R.Sherman, and D.V.Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans Networking*, 5, 1997.

[11] L. Xu and I. Rhee. CUBIC: A new TCP-Friendly high-speed TCP variant. In *Proc. Workshop on Protocols for Fast Long Distance Networks, 2005*, 2005.

[12] Y.Lee, D. Leith, and R.N.Shorten. Experimental evaluation of TCP protocols for high-speed networks. Accepted for publication in IEEE Transactions on Networking, 2007.