A Comparative Performance Evaluation of DCCP

Saleem Bhatti
School of Computer Science
University of St Andrews
St Andrews
Fife KY16 9SX UK
email: saleem@cs.st-andrews.ac.uk

Martin Bateman
School of Computer Science
University of St Andrews
St Andrews
Fife KY16 9SX UK
email: mb@cs.st-andrews.ac.uk

Dimitris Miras
Department of Computer Science
UCL
London WC1E 6BT UK
email: d.miras@cs.ucl.ac.uk

***Abstract*—Interest continues to grow in alternative transport protocols to the Transmission Control Protocol (TCP). These alternatives include protocols designed to give greater efficiency in high-speed, high-delay environments (so-called *high-speed TCP variants*), and protocols that provide congestion control without reliability. For the former category, along with the deployed base of 'vanilla' TCP – TCP NewReno – the TCP variants BIC and Cubic are widely used within Linux: for the latter category, the Datagram Congestion Control Protocol (DCCP) is currently on the IETF Standards Track. It is clear that future traffic patterns will consist of a mix of flows from these protocols (and others). So, it is important for users and network operators to be aware of the impact that these protocols may have on users. We assess the performance of DCCP CCID2 relative to TCP NewReno, and variants BIC and CUBIC, all in "out-of-the box" configurations. We use a testbed and end-to-end measurements to assess overall throughput, and also to assess fairness – how well these protocols might respond to each other when operating over the same end-to-end network path. We find that DCCP CCID2 shows good fairness with NewReno under our test conditions, while BIC and CUBIC show unfairness above round-trip times of 25ms.**

I. Introduction

While the Transmission Control Protocol (TCP) remains in widespread use, new transport protocols are being defined with different behaviour to that of TCP. TCP's additive increase multiplicative decrease (AIMD) behaviour [1] is often credited as a major factor in the stability of today's Internet. The AIMD behaviour causes TCP to back-off when it experiences congestion, cutting its transmission rate to half, and then only increasing its transmission rate by one segment size every round-trip-time (RTT). However, this 'standard' TCP behaviour has performance problems when considering certain scenarios:

- TCP's back-off behaviour for congestion avoidance and control is considered conservative and results in poor utilisation in networks with paths that have large delays and/or high end-to-end capacity, i.e. paths with a high bandwidth-delay product (BDP). In such circumstances, available network capacity is underused.
- TCP's behaviour is based on the requirement of reliable delivery (through retransmission). However, some applications may not need reliability, whilst still requiring congestion control, e.g. media streaming, sensor networks, online gaming, high-capacity data streams from streams applied science applictaions (e.g. Grid applications).

For the first of these cases, many *high-speed TCP variants* have been implemented with different behaviour, which allows them to be more effective on high-BDP paths. Indeed, for Linux, two of these variants, BIC [2] and CUBIC [3], are in common use, and are the default versions of TCP in place of NewReno in Linux kernels over the past few years[1].

For the second of these cases, the Datagram Congestion Control Protocol (DCCP) [4] has emerged as the likely protocol to provide congestion controlled transport service to applications, without the reliability of TCP. DCCP is on the IETF standards track, and an implementation is now available within the Linux kernel.

Both BIC and CUBIC are designed to be more "aggressive" than NewReno, in order to make use of under-utilised capacity. Our recent results show that this is indeed the case for high-BDP paths at $\sim$1Gb/s [5], supporting results in similar studies at sub-gigabit data rates [6], [7].

While NewReno, BIC and CUBIC are in widespread use in Linux, as DCCP matures and its use increases, it is important for users to be aware of its behaviour within an environment where there will be a mix of protocols in operation. *What happens when DCCP flows share an end-to-end path with NewReno, BIC or CUBIC flows?*

Our paper first introduces the methodology and metrics we will use in order to answer the question above, including a description of our testbed (Section II). We then describe the behaviour of the individual protocols as observed on our testbed (Section III), before examining the inter-protocol interactions (Section IV). We then conclude and list items for future work (Section V).

II. Methodology and Metrics

We have taken a practical approach, generating data flows using a modified version of the tool *iperf* [2], our modifications allow reporting of additional TCP state information[3]. We transmitted flows over a simple testbed, sending two flows over a single bottelneck link. Our intention was to make observations of the end-to-end behaviour of the flows over the

[1]NewReno before kernel version 2.6.8, August 2004; BIC from Linux kernel version 2.6.8, August 2004; CUBIC from Linux kernel version 2.6.19, September 2006.

[2]http://dast.nlanr.net/Projects/Iperf/

[3]This modified version of *iperf* is available from the authors.

bottleneck link, and measure their relative performance. We based our evaluation on the end-to-end throughput achieved by each flow, as reported at 1s intervals by *iperf*. We used the throughput measurements to evaluate how *fair* the resource share was for the two flows.

A. *Testbed*

Our testbed[4] set-up was the well-known dumbbell arrangement as depicted in Figure 1 and used in previous similar studies [6]–[8], albeit at higher speeds (100s of Mb/s to nearly $\sim$1Gb/s). This simple testbed helps to reduce the factors of error or unknown behaviour that may affect the results and concentrate on the protocol behaviour. As noted in [9], "Simple topologies, like a "dumbbell" topology with one congested link, are sufficient to study many traffic properties."

The testbed consisted of two senders, two receivers and a router to provide the network delay and bottleneck. All network connections were 100Mb/s full-duplex Ethernet. Our measurement runs consisted of generating two flows, using *iperf*, for a pair-wise comparison: Flow 1 was from Sender 1 to Receiver 1, and Flow 2 was from Sender 2 to Receiver 2. The duration of each measurement run was 300s, with Flow 1 starting at 0s, and Flow 2 starting at 30s to avoid initial synchronisation effects. After some calibration tests, we conducted five measurement runs for each of the TCP protocols running against DCCP/CCID2 (CCID2 is explained later) at each of the following RTT values: 25ms, 50ms, 75ms, 100ms, 125ms, 150ms, 175ms, 200ms. So, in all these cases, TCP's normal 64KB window is lower than the BDP of the path.

The senders and receivers ran Linux kernel version 2.6.22.6, and we used "out-of-the-box" configuration for the end systems[5], rather than tuning the stack for high-speed operation (as in [6]–[8]), in order to gauge the performance under (arguably) the most likely configuration of the end-system. TCP Selective Acknowledgements (SACK, RFC2018) was enabled; the Window Scale Option, Protection Against Wrapped Sequence Numbers, and Round-Trip Time Measurement (RFC1323) were enabled; and MTU size was 1500 bytes (no IP fragementation): all these being default settings.

The netem router ran Linux kernel version 2.6.18 and the package *netem*[6] was used to control RTT for the packet flows, with the network delay split equally between the forward and reverse paths. Buffer sizes on the router were set to ensure enough buffer space to handle the window sizes of the end-system TCP stacks, i.e. the buffering on the router was always set to 100% of the BDP for the given RTT.

To check the behaviour of our testbed set-up, we used *ping* to measure the RTT that was configured at the *netem* router: *ping* reported the delay was accurately configured (to within $\sim$1ms). We then generated single TCP flows and observed that all TCP variants and DCCP/CCID2 reached peaks of $\sim$95Mb/s at the lower RTTs (within the normal 64KB window of TCP) – example runs showing end-to-end throughput at various RTT values are shown in Figures 7(a), 8(a), 9(a) and 10(a). We also generated two flows using the same protocol, to gauge how fair the protocol was to itself – example runs showing end-to-end throughput for pair-wise tests are given in Figures 7(b), 8(b), 9(b) and 10(b).

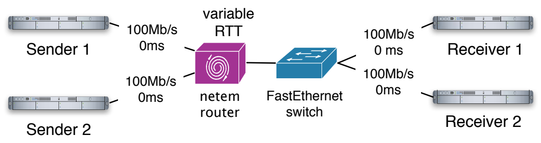


Figure 1. Testbed configuration

B. *Fairness*

We have used a simple and practical methodology to assess inter-flow fairness, based on the use of a testbed and measurements of throughput. The Transport Modeling Research Group of the IRTF[7] (TMRG) is currently formulating criteria by which one might provide assessments, notably comparative assessments, of transport protocols. The TMRG notes that one metric to be assessed is *inter-flow fairness:* how are resources shared between different flows?

A metric used widely for assessing fairness is Jain's Fairness Index (JFI) [10] as in (1), where, $0 \leq J \leq 1$, N is the number of flows, r_n is the value of the resource attribute being assessed for flow n, e.g. r_n is the measured end-to-end throughput. $J = 1$ means there is fairness across all flows; $J = 0$ indicates no fairness.

JFI gives an evaluation of *system-wide* behaviour, over a given time-span, i.e. the behaviour considering all flows. An obvious approach to examining system-wide fairness over time is simply to evaluate the JFI at given instants during the period of interest, as in (2), where t, in practise, will be discrete. So, $r_n(t_j)$ is an approximation of throughput as determined at time interval t_j, and evaluated over the period $(t_j, t_{j-1}), t_j > t_{j-1}$, where t_{j-1} is the previous time at which an approximation was determined. For our experiments, this was once every second. So, J is the mean value of $J(t)$ values over a given time period.

$$J = \frac{\left(\sum_{n=1}^{N} r_n\right)^2}{N \sum_{n=1}^{N} r_n^2} \qquad (1)$$

$$J(t) = \frac{\left(\sum_{n=1}^{N} r_n(t)\right)^2}{N \sum_{n=1}^{N} r_n(t)^2} \qquad (2)$$

We configured *iperf* to report throughput values at 1s intervals, and evaluated $J(t)$ for each throughput measurement. Flow 2 started at $t = 30s$, so we used values from $t = 60s$

[4]Full details, including hardware specification and Linux kernel parameter settings are available from the authors.

[5]DCCP/CCID2 was configured as recommended in http://www.linux-foundation.org/en/Net:DCCP.

[6]http://www.linux-foundation.org/en/Net:Netem

[7]http://www.icir.org/tmrg/

(to permit Flow 2 to stabilise) to $t = 300$ (the end of the measurement run) in order to calculate values for JFI.

While discussions continue within the community on how to assess fairness, e.g. by use of end-to-end delay for file transfers (e.g. [11]), or some notion of 'cost' (e.g. [12]), we choose to use end-to-end throughput, as it remains widely used, is easily understood and straightforward to measure.

III. Protocol behaviour

In this section, we describe briefly the behaviour of the protocols we will consider. Our description is intended to highlight the main features of each protocol. We include graphs of throughput from the output of our testbed calibration tests:

- for an individual flow, showing their respective utilisation on our testbed set-up, demonstrating the end-to-end throughput each protocol is capable of when it is the only flow on the testbed.
- for two flows, showing that each protocol is capable of adapting its behaviour in the presence of another flow of the same type and resulting in convergence to a fair share of the available end-to-end capacity.

A. TCP NewReno

The basic congestion control algorithm in TCP NewReno is well known [1]. To control transmission, a *congestion window (cwnd)*, is subject to Additive Increase Multiplicative Decrease (AIMD) behaviour:

$$\begin{aligned} OnACK : cwnd &\leftarrow cwnd + \alpha \\ OnLoss : cwnd &\leftarrow \beta.cwnd \end{aligned}$$

with $\alpha = 1$ and $\beta = 0.5$. The value of $cwnd$ increases by α segments when an ACKnowledgment is received, and decreases a factor β when a loss is detected. The other TCP variants typically use different algorithms to reduce and increase the window size, and so control the rate of transmission. In Figure 7(a), we note that TCP takes longer to achieve higher throughput as the RTT increases.

B. BIC

Binary Increase Congestion control TCP – *BIC* [2] – uses a binary search algorithm between the window size just before a reduction (W_{max}) and the window size after the reduction (W_{min}). If w_1 is the midpoint between W_{min} and W_{max}, then the window is rapidly increased when it is less than a specified distance, S_{max}, from w_1, and grows more slowly when it is near w_1. If the distance between the minimum window and the midpoint is more then S_{max}, the window is increased by S_{max}, following a linear increase. BIC reduces $cwnd$ by a multiplicative factor β. If no loss occurs, the new window size becomes the current minimum, otherwise, the window size becomes the new maximum. If the window grows beyond the current maximum, an exponential and then linear increase is used to probe for the new equilibrium window size. In Figure 8(a), compared to TCP (Figure 7(a)), BIC achieves higher throughput and more quickly at larger RTT values.

C. CUBIC

CUBIC [3] uses a cubic function to control its congestion window growth. If W_{max} is the congestion window before a loss event, then after a window reduction, the window grows fast and then slows down as it approaches W_{max}. Around W_{max}, the window grows slowly, again accelerating as it moves away from W_{max}. The following formula determines the congestion window size ($cwnd$):

$$cwnd = C(T - K)^3 + W_{max}$$

where C is a scaling constant, T is the time since the last loss event and $K = \sqrt[3]{W_{max}\frac{\beta}{C}}$, where β is the multiplicative decrease factor after a loss event. C and β are set to 0.4 and 0.2 respectively. To increase fairness and stability, the window is clamped to grow linearly when it is far from W_{max}.

The behaviour of CUBIC seems very similar to BIC: Figure 8 (BIC) with compare Figure 9 (CUBIC).

D. DCCP/CCID2

The Datagram Congestion Control Protocol (DCCP), " ... is a transport protocol that provides bidirectional unicast connections of congestion-controlled unreliable datagrams. DCCP is suitable for applications that transfer fairly large amounts of data and that can benefit from control over the tradeoff between timeliness and reliability." [4]. DCCP allows different flow adaptation mechanisms – *profiles* – for congestion control to be defined and used. For example, the profile designated "Congestion Control ID 2" (CCID2) [13] is defined to be "TCP-like congestion control", i.e. as close as possible to the AIMD behaviour described in Section III-A[8]. It is to be noted that, at this time, the Linux implementation of DCCP/CCID2 is a work-in-progress, but relatively stable.

We note that on visual inspection, the behaviour of DCCP/CCID2 is closer to that of NewReno than to either BIC or CUBIC, and this is to be expected from its design. In Figure 10(a), we note that although the pattern of throughout is similar to NewReno (Figure 7(a)), DCCP/CCID2 does not behave exactly the same, again to be expected from its design.

E. Self Fairness

When two flows of the same protocol are transmitted across the same path at the same time, we see that they achieve fairness. In Figures 7(b), 8(b), 9(b) and 10(b) we see examples of 2 flows and can see that for each respective protocol, there appears to be good fairness. Table I shows the mean value of JFI from Equation (2), taken as a mean value over five runs, for each RTT, confirming good fairness of each protocol against another flow of the same protocol. Mean throughput values are shown in Table II.

[8]Other CCIDs are also being defined: CCID3, "TCP Friendly Rate Control (TFRC)" is also implemented in Linux.

	RTT (ms) [BDP (MB)]				Example
	25 [0.31]	50 [0.62]	100 [1.25]	200 [2.50]	Figure Thr'put
NewReno	0.96	0.96	0.97	0.99	7(b)
BIC	0.97	0.94	0.96	0.88	8(b)
CUBIC	0.97	0.95	0.94	0.86	9(b)
CCID2	0.96	0.95	0.98	0.94	10(b)

Table I
JFI VALUES FOR TWO FLOWS OF THE SAME PROTOCOL (MEAN OVER 5 RUNS, FROM $t = 60s$ TO $t = 300s$)

	RTT (ms)							
	25		50		100		200	
Flow	1	2	1	2	1	2	1	2
NewReno	45	49	42	51	43	51	43	43
BIC	43	51	53	41	44	49	42	41
CUBIC	42	52	51	43	44	48	41	41
CCID2	43	51	46	47	45	48	33	32

Table II
THROUGHPUT VALUES (MB/S) FOR TWO FLOWS OF THE SAME PROTOCOL (MEAN OVER 5 RUNS, FROM $t = 60s$ TO $t = 300s$)

IV. INTERPROTOCOL BEHAVIOUR

We now look at interaction of our chosen protocols by examining the behaviour of two flows across the testbed. Flow 1 was started at 0s and was a DCCP/CCID2 flow. Flow 2 was started at 30s and was one of TCP NewReno, BIC, or CUBIC. We executed five runs of each pair-wise experiment for each of the RTT values as explained in Section II-A. We recorded the value of end-to-end throughput reported by *iperf* at 1s intervals from $t = 60s$ (allowing 30s for Flow 2 to stabilise) to $t = 300s$ (the end of the experimental run). The throughput values were used with Equation (1), to assess a summary of system behaviour; and with Equation (2) to examine visually the dynamics of the interaction over the duration of the experiment. We also noted the mean throughput of the two flows over the five runs.

As a summary of the behaviour, the values of JFI generated using (1) are given in Table III (the mean JFI values and standard deviations over five runs) and shown in Figure 2. Table V gives the mean throughputs over five runs for each protocol, where Flow 1 in the table is always DCCP/CCID2 and Flow 2 is the TCP variant given in the first column of that row. We note that fairness of DCCP/CCID2 and TCP NewReno is good across the range of RTT values examined, and this is encouraging as it meets a goal of CCID2 to be "TCP-like". However, our initial observation is that BIC and CUBIC have poor fairness with CCID2 beyond an RTT value of 25ms, and the unfairness is due to the TCP variant using more capacity than DCCP/CCID2. BIC and CUBIC have similar behaviour.

This behaviour is to be expected: as the RTT (and so the BDP) increases, from examining the performance of the individual flows (from Section III), it can be seen that DCCP/CCID2 becomes increasingly poor at utilising the available capacity (Figure 10(a)), whilst BIC (Figure 8(a)) and CUBIC (Figure 9(a)), have better performance at higher BDP, as per their design. So, the unfairness observed in Figure 2 may not necessarily be a concern. Note, however, that as the RTT increases beyond ∼125ms, the fairness improves as CCID2 throughput improves. We believe that this is because the larger window size at the higher BDP has a more significant effect on the end-to-end throughput than the more aggressive behaviour of BIC and CUBIC, though we have not yet examined this explicitly.

When we re-run the experiments with the TCP flavour starting first (flow 1) and then DCCP/CCID2 (flow 2), we obtain the results given in Figure 3 and Table IV. We note that the behaviour for BIC and CUBIC is quite different from that in Figure 2. There is still unfairness with DCCP/CCID2 but the effect is not significant until 100ms, and then drops sharply, and does not improve, as it does when the DCCP/CCID2 flow is started first. Again, this behaviour needs to be investigated further in future work. The rest of this paper focuses on the first set of results in Figure 2, Table III and Table V, in which we see that unfairness starts after 25ms, which can be considered to represent a worse case compared to Figure 3.

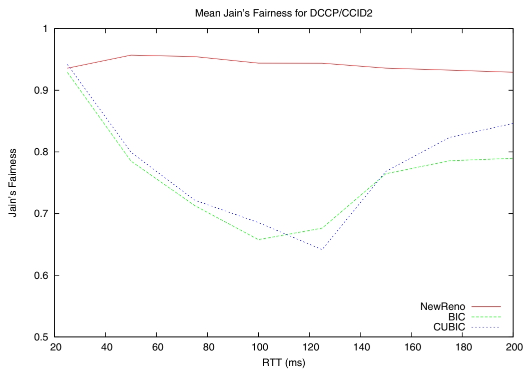


Figure 2. Pair-wise behaviour, DCCP/CCID2 flow first (See Table III)

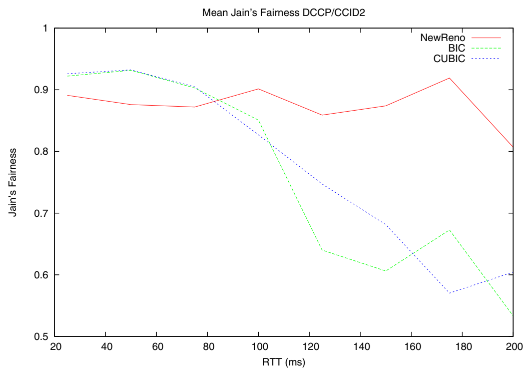


Figure 3. Pair-wise behaviour, TCP flow first (See Table IV)

Now, we examine separately, each pair-wise experiment. To show the dynamics of the interaction for each protocol against DCCP/CCID2, we have plotted over the duration of one of the runs, the JFI values as in Equation (2). We have shown only an example run at the RTT values of 25ms, 50ms, 100ms and 200ms for the sake of clarity on the graphs.

	RTT (ms) [BDP (MB)]			
	25 [0.31]	50 [0.62]	75 [0.93]	100 [1.25]
Reno	0.94, 0.05	0.96, 0.04	0.95, 0.05	0.94, 0.05
BIC	0.93, 0.06	0.79, 0.09	0.71, 0.07	0.66, 0.09
CUBIC	0.94, 0.05	0.80, 0.09	0.72, 0.09	0.69, 0.08
	125 [1.57]	150 [1.88]	175 [1.19]	200 [2.50]
Reno	0.94, 0.05	0.94, 0.08	0.93, 0.07	0.93, 0.06
BIC	0.68, 0.06	0.76, 0.06	0.79, 0.05	0.79, 0.06
CUBIC	0.64, 0.11	0.77, 0.06	0.82, 0.06	0.85, 0.05

Table III
⟨JFI values, standard deviation⟩ for pair-wise tests against DCCP/CCID2, Flow 1 = DCCP/CCID2, Flow 2 = TCP (mean over 5 runs, from $t = 60s$ to $t = 300s$) (See Figure 2)

	RTT (ms) [BDP (MB)]			
	25 [0.31]	50 [0.62]	75 [0.93]	100 [1.25]
Reno	0.89, 0.01	0.88, 0.01	0.87, 0.01	0.9, 0.02
BIC	0.92, 0.00	0.93, 0.01	0.90, 0.01	0.85, 0.03
CUBIC	0.93, 0.01	0.93, 0.01	0.90, 0.02	0.83, 0.04
	125 [1.57]	150 [1.88]	175 [1.19]	200 [2.50]
Reno	0.86, 0.02	0.87, 0.02	0.92, 0.03	0.81, 0.04
BIC	0.64, 0.14	0.61, 0.17	0.67, 0.16	0.53, 0.18
CUBIC	0.75, 0.07	0.68, 0.10	0.57, 0.14	0.60, 0.15

Table IV
⟨JFI values, standard deviation⟩ for pair-wise tests against DCCP/CCID2, Flow 1 = TCP, Flow 2 = DCCP/CCID2 (mean over 5 runs, from $t = 60s$ to $t = 300s$) (See Figure 3)

A. DCCP/CCID2 vs NewReno

As might be expected, there is good fairness between DCCP/CCID2 and NewReno, with $J > 0.90$, and as seen in Table III and visually in Figure 4. Various differences in the CCID2 behaviour, however, mean that it will not perform as well as NewReno. In Table V, we can see that the throughput of DCCP/CCID2 (Flow 1) is better at lower RTTs, but starts to get slightly worse at higher RTTs (above 75ms). Section 3.1 of [13] summarises the similarities between SACK-based TCP (as NewReno is) and DCCP/CCID2:

	RTT (ms)							
	25		50		75		100	
Flow	1	2	1	2	1	2	1	2
NewReno	55	40	52	42	46	48	41	50
BIC	40	54	26	67	17	74	15	74
CUBIC	40	54	25	68	19	73	16	74
	125		150		175		200	
Flow	1	2	1	2	1	2	1	2
NewReno	39	49	37	39	31	33	32	36
BIC	16	74	22	71	23	71	23	70
CUBIC	13	75	22	70	26	68	27	67

Table V
Throughput (Mb/s) for pair-wise tests against DCCP/CCID2 (mean over 5 runs, from $t = 60s$ to $t = 300s$)

- DCCP/CCID2 uses a close variant of the AIMD behaviour in TCP, including window halving and linear congestion avoidance, using the variables $cwnd$ and $ssthresh$.
- DCCP/CCID2 uses a close variant of SACK, employing an *Ack Vector* which contains the same information that might be found in the TCP SACK option.
- *DCCP-Ack* packets are used to measure round trip time (including the option for a Timestamp as in TCP), and "clock out" the data from the sender.

However, there are some differences which will have some effect on the control of transmissions from the sender:

- DCCP applies congestion control to the DCCP-Ack packets it generates at the receiver. For an *Ack Ratio* of R, DCCP-Acks are generated every R packets that are received. This could, potentially, slow down the rate of acknowledgements for a number of reasons, e.g. loss/delay of a DCCP-Ack has a greater effect on the sender when R is high, compared to TCP which does not use congestion control on generation of ACKs. Of course, this will depend also on packet loss rates.
- TCP is a byte-stream protocol whilst DCCP is datagram based so variables such as $cwnd$ and $ssthresh$ used to control transmissions, which my have units of bytes in TCP, are measured in datagrams in DCCP. In operations, especially at high BDP with large window sizes, this is likely to become insignificant compared to the rate of ACKs, unless the datagram size is very large (e.g. if using a large MTU).
- As DCCP does not use retransmissions, TCP fast-recovery mechanisms are not implemented. Again, this is likely to be an insignificant factor compared to the rate of DCCP-Acks, unless operating in an environment where there are high loss rates, e.g. due to high congestion or high bit error rates.

If we examine Figure 4, we see that the fairness is stable across two flows for all RTTs (little variation in the values of JFI over time). In Table III, we see that as well as the JFI value being high, the standard deviation is small and stable, increasing slightly at 150ms and 175ms. In the example plot of $J(t)$ values given in Figure 4, this is seen clearly as low variation in the values of $J(t)$ across all RTTs.

B. DCCP/CCID2 vs BIC & CUBIC

Since we found that the behaviour of BIC and CUBIC is similar, we discuss them together in this subsection.

In the case of both BIC & CUBIC run against DCCP/CCID2, we see from Figure 2 and Table III that the JFI value is below 0.8 at all values of RTT above 25ms, i.e. there would appear, at first sight, to be a great deal of unfairness between BIC & CUBIC and DCCP/CCID2. In Table V, we see that BIC & CUBIC always has a higher throughput than DCCP/CCID2, at all our RTT values.

However, it is not sufficient just to consider the JFI of the pair-wise tests in order to make a true assessment of fairness. We must also consider that BIC/CUBIC are designed for high BDP paths, whereas DCCP/CCID2 is designed to exhibit "TCP-like" behaviour. So, for BIC, CUBIC and DCCP/CCID2, respectively, we must take into account the likely throughputs under circumstances where they may be competing equally with the other flow in the experimental run. That is, we need to make an assessment of whether or not DCCP/CCID2 is actually being constrained by the more aggressive behaviour of BIC/CUBIC. *Is the higher throughput of the BIC & CUBIC flows due simply to those protocols using the capacity that DCCP/CCID2 is not able to use effectively at high BDPs?*

We can answer this question by comparing the mean throughput figures in Table V (which records the mean throughputs of the *pair-wise* experiments) and Table II (which records the mean throughputs of *two flows* of the same protocol). So, we are trying to assess fairness not just by looking at the JFI values, but also by comparing the protocol performance of DCCP/CCID2 against a protocol with which it has very fair behaviour at all RTTs, i.e. itself.

Let us first consider BIC at RTT=25ms. Recall that in our pair-wise experiments, Flow 1 is always DCCP/CCID2. So, let us compare the Flow 1 values in Table II for CCID2, and the values at the same RTT in Table V. We find that at RTT=25ms, the throughputs of both DCCP/CCID2 and BIC are similar: DCCP/CCID2 (Table II Flow 1) is 43Mb/s, and BIC (Table II Flow 2) is 51Mb/s; DCCP/CCID2 (Table V Flow 1) is 40Mb/s, and BIC (Table V Flow 2) is 54Mb/s. So, there is very good fairness, arguably even better than the value of JFI=0.93 suggests (from Table III).

Let us now consider BIC at RTT=100ms. We find that the throughputs of both DCCP/CCID2 and BIC are very different: DCCP/CCID2 (Table II Flow 1) is 45Mb/s, and BIC (Table II Flow 2) is 49Mb/s; DCCP/CCID2 (Table V Flow 1) is 15Mb/s, and BIC (Table V Flow 2) is 74Mb/s. It seems clear that in this case, the "non-TCP-like" behaviour of BIC has a detrimental effect on the end-to-end throughput of DCCP/CCID2: BIC is being more aggressive than DCCP/CCID2 and causing unfairness, arguably worse than the JFI value of 0.66 might reflect, as the DCCP/CCID2 throughput is approximately a third of what it would be against another DCCP/CCID2 flow.

A similar situation is observed when comparing JFI values and throughput values for CUBIC.

Also, for both BIC and CUBIC, there is greater instability in the flow throughput: the respective pair-wise experiments of CCID2-BIC and CCID2-CUBIC show that flows are more bursty at all RTTs when compared to CCID2-NewReno – compare the values of standard deviation in Table III; and visually in Figures 5 (for BIC) and 6 (for CUBIC) with Figure 4 (for NewReno).

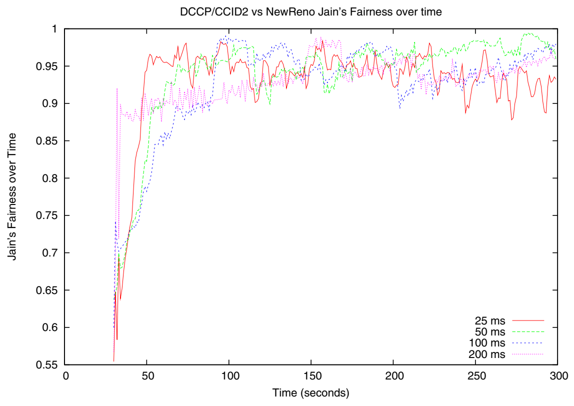


Figure 4. NewReno: Typical JFI values over a single run.

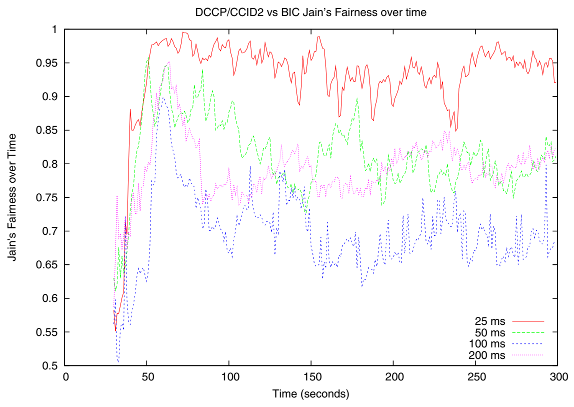


Figure 5. BIC: Typical JFI values over a single run.

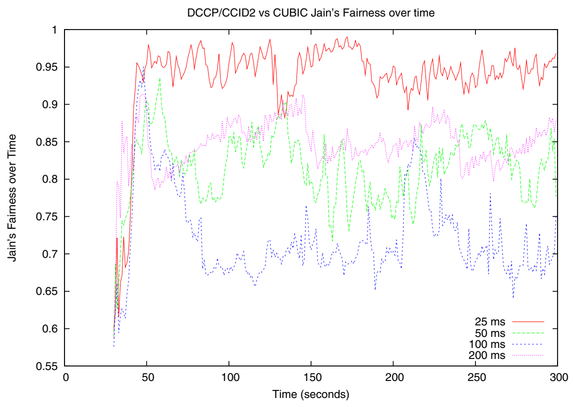


Figure 6. CUBIC: Typical JFI values over a single run.

V. Conclusions and Future Work

We have examined the interaction between DCCP/CCID2, which is currently being implemented and deployed, and variants of TCP that are already widely used: TCP NewReno, BIC and CUBIC. We have observed that, with link speeds of 100Mb/s and at RTTs between 25ms and 200ms, DCCP/CCID2 has good fairness with TCP NewReno at all RTT values, though NewReno does achieve a slightly higher throughput. BIC and CUBIC, always achieved higher throughputs in our tests, becoming unfair above RTT=25ms, with greatest unfairness at $\sim$125ms, after which, fairness improves as the RTT increases. So, in environments where the normal 64KB TCP window is exceed, where DCCP/CCID2 is sharing an end-to-end path with mainly TCP NewReno flows, it is likely that there will be good fairness to DCCP flows. With BIC and/or CUBIC, DCCP/CCID2 will achieve lower utilisation as these protocols have been observed to show lower fairness to DCCP/CCID2 in our tests, being more aggressive at higher BDPs than DCCP/CCID2. Also, for BIC & CUBIC we observed greater burstiness of flows between 50ms and 125ms, compared with the situation when DCCP/CCID2 shares an end-to-end path with NewReno.

A. Thoughts for future work

The “dip” in fairness as observed in Figure 2 needs to be investigated. It is not clear from our experiments why the fairness decreases (as one might expect) but then increases, with increasing RTT. Also, does this trend continue at higher BDP, e.g. at link speeds of 1Gb/s and/or higher RTT values?

It can be seen that as we approach the higher RTT values, the fairness is improving for BIC and CUBIC in Figure 2 but not in Figure 3. It would be informative to investigate the reason for this difference in behaviour. Also, it would be informative to observe the flow dynamics at higher BDP values, for example at higher speeds such as several 100 Mb/s or 1Gb/s. Such high speeds may well be an application domain for DCCP as desktop/access speeds improve, and more demanding applications begin to use DCCP, for example the media streaming and Grid applications mentioned in the introduction.

We are also planning tests using Microsoft Windows TCP stacks (XP and Vista).

DCCP/CCID2 can also operate using Explicit Congestion Notification (ECN). Would use of ECN result in an improvement in throughput or fairness?

We observe that whilst Jain’s Fairness Index (JFI) is widely used for comparing protocol performance, JFI assumes that all processes can make equal use of the available resource: it is currently not defined to be able to capture weightings of values used in order to express relative capability. So, it would be beneficial to investigate modification of the JFI, or definition of new metrics that may be able to provide such functionality.

References

[1] V. Jacobson and M. Karels, “Congestion avoidance and control,” in *ACM SIGCOMM*, Aug 1988.

[2] L. Xu, K. Harfoush, and I. Rhee, “Binary increase congestion control for fast long-distance networks,” in *IEEE INFOCOM*, Mar 2004.

[3] L. Xu and I. Rhee, “CUBIC: A new TCP-Friendly high-speed TCP variant,” in *Third International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)*, Feb 2005.

[4] E. Kohler, M. Handley, and S. Floyd, “Datagram Congestion Control Protocol (DCCP),” RFC 4340 (Proposed Standard), Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[5] D. Miras, M. Bateman, and S. Bhatti, “Fairness of High-Speed TCP Stacks,” in *Proc. AINA2008 - IEEE 22nd International Conference on Advanced Information Networking and Applications*, March 2008.

[6] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu, “A Step Toward Realistic Performance Evaluation of High-Speed TCP Variants,” in *Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)*, Feb 2006.

[7] Y.-T. Li, D. Leith, and R. N. Shorten, “Experimental Evaluation of TCP Protocols for High-Speed Networks,” to appear. [Online]. Available: http://www.hamilton.ie/net/eval/ToNfinal.pdf

[8] M. Bateman, S. Bhatti, G. Bigwood, D. Rehunathan, C. Allison, T. Henderson, and D. Miras, “A Comparison of TCP Behaviour at High Speeds Using ns-2 and Linux,” in *CNS2008 - 11th Communications and Networking Simulation Symposium*, April 2008.

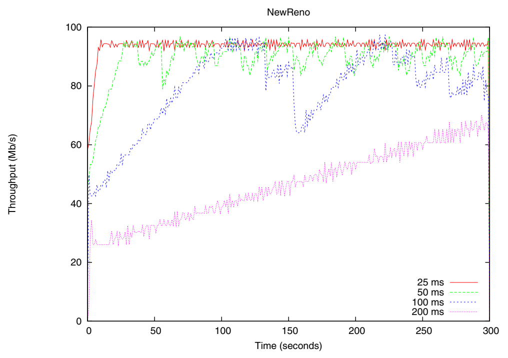
[9] S. Floyd and E. Kohler, “Internet research needs better models,” in *Proceedings of HotNets–I*, October 2002. [Online]. Available: citeseer.ist.psu.edu/floyd02internet.html

[10] D.-M. Chiu and R. Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, 1989.

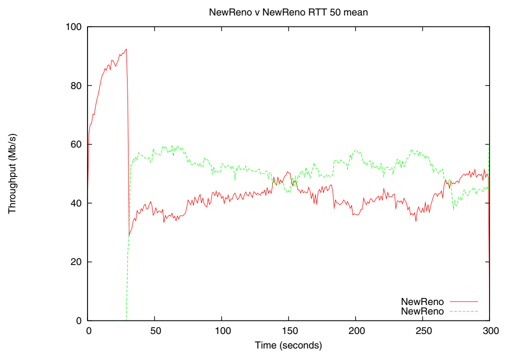
[11] S. Kunniyur and R. Srikant, “End-to-end Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 689–702, October 2003.

[12] B. Briscoe, “Flow rate fairness: dismantling a religion,” *SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 63–74, 2007.

[13] S. Floyd and E. Kohler, “Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control,” RFC 4341 (Proposed Standard), Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4341.txt

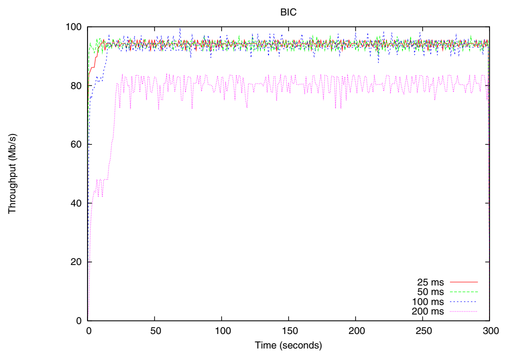


(a) Singe flow, at various RTT

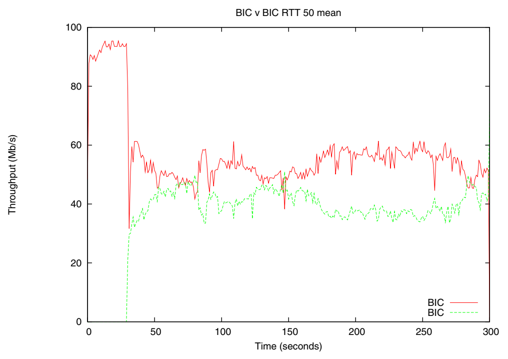


(b) Two flows, RTT=50ms

Figure 7. NewReno: typical behaviour on our testbed

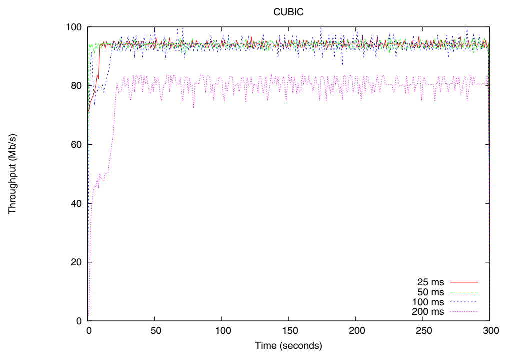


(a) Singe flow, at various RTT

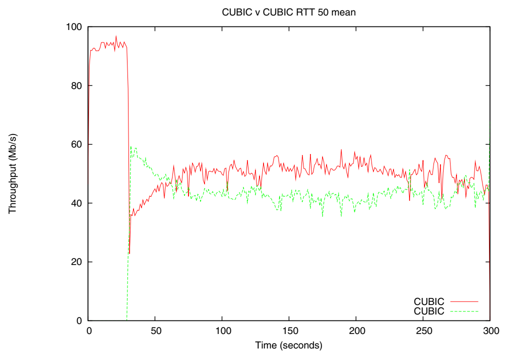


(b) Two flows, RTT=50ms

Figure 8. BIC: typical behaviour on our testbed

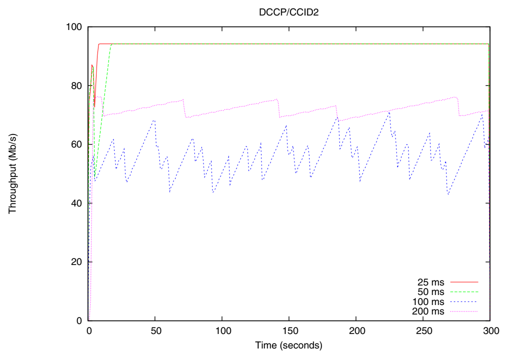


(a) Singe flow, at various RTT

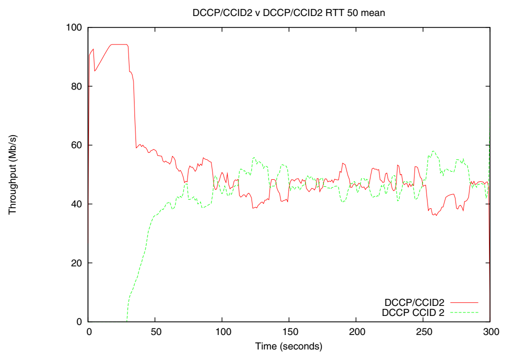


(b) Two flows, RTT=50ms

Figure 9. CUBIC: typical behaviour on our testbed



(a) Singe flow, at various RTT



(b) Two flows, RTT=50ms

Figure 10. DCCP/CCID2: typical behaviour on testbed