Transport Protocol Throughput Fairness

S. Bhatti, M. Bateman
School of Computer Science, University of St Andrews, St Andrews, UK
Email: {saleem, mb}@cs.st-andrews.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 show the measurement of fairness in throughput performance of DCCP Congestion Control ID 2 (CCID2) relative to TCP NewReno, and variants Binary Increase Congestion control (BIC), CUBIC and Compound, 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, in our testbed, DCCP CCID2 shows good fairness with NewReno, while BIC, CUBIC and Compound show unfairness above round-trip times of 25ms.**

Index Terms— **fairness, protocol performance, DCCP, TCP, BIC, CUBIC, Compound TCP**

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 applied science applications (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, Binary Increase Congestion control (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]. In Windows, as well as TCP NewReno, Compound TCP [4] is being introduced.

For the second of these cases, the Datagram Congestion Control Protocol (DCCP) [5] 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.

A. Fairness in throughput performance

BIC, CUBIC and Compound TCP 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 BIC and CUBIC in high-BDP paths at ∼1Gb/s [6], supporting results in similar studies at sub-gigabit data rates [7], [8].

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. Of course, Windows is the most widely used desktop platform, so, potentially, the use of Compound TCP will also become widespread. The question of fairness then arises: *What happens when DCCP flows share an end-to-end path with NewReno, BIC, CUBIC or Compound TCP flows?*

This paper is based on two items of previous work: (1) "A Comparative Performance Evaluation of DCCP", by S. Bhatti, M. Bateman, and D. Miras, which appeared in the Proceedings of *SPECTS2008 - 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, University of Edinburgh, UK. 16-18 June 2008; (2) "Revisiting inter-flow fairness", by S. Bhatti, M. Bateman, D. Rehunathan, T. Henderson, G. Bigwood, D. Miras, which appeared in the Proceedings of *BROADNETS2008 - 5th International Conference on Broadband Communications, Networks and Systems*, London, UK 08-11 September 2008.

[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.

To answer this questions, we need to ask ourselves, "How can we assess 'fairness' in the behaviour of such protocols?" A popular measure of fairness for network flows is *Jain's Fairness Index (JFI)* [9]. While this is a general metric, by using the end-to-end throughput of the flows sharing (whole or part of) a network path, system-wide (i.e., across all flows) fairness can be assessed. Starting with Jain's Fairness Index, then introducing a new metric – *Normalised Resource Usage (NRU)* – we propose a simple but practical methodology for examining the throughput fairness of flows.

B. Contribution and structure of this paper

The contributions of this paper are:

- a new metric with which to assess the dynamics of inter-protocol fairness with respect to throughput.
- a measurement-based approach which allows the new metric to be used easily.
- a measurement-based, experimental pair-wise assessment of the fairness of NewReno, BIC, CUBIC and Compound TCP against DCCP operating with a TCP-like congestion control (CCID2) [10].

We begin with a discussion on the definition of 'fairness' in Section II. Based on this discussion, in Section III, we then define our metrics and methods for evaluation. For rigour and clarity, we present our results first for individual protocol behaviour in Section IV, and then for the inter-protocol behaviour in Section V. In Section VI, we present our conclusion, including a discussion of the potential limitations of our approach.

II. Assessing inter-flow fairness

The Transport Modelling Research Group (TMRG) of the IRTF[2] presents the criteria by which one might make rigourous and complete assessments, notably comparative assessments, of transport protocols [11]. One of the measures noted by the TMRG as a desirable metric for assessing the performance of transport flows is for inter-flow fairness: sharing of resources between different flows.

Our intention is to demonstrate a methodology for measuring inter-flow fairness that is easy to implement. We show how fairness, as evaluated using end-to-end performance measurements (we chose throughput), can be utilised for assessing relative performance.

A. What is fair?

The notion of 'fairness' in the use of resources has been much debated within the literature. Having a fair share of a resource is important where the resource demands of multiple flows sharing the same resource are not met. In the absence of any other resource controls in the network, this means that there is at least one point along the end-to-end path where congestion is occurring, and we may determine how the resource is being shared by evaluating the resource distribution across the flows on that (part of) the path. For example, in the case of transport protocol flows sharing a bottleneck link on an end-to-end path, we could evaluate the way that the capacity is shared at the bottleneck (a local view), or the end-to-end throughput achieved for each flow sharing the bottleneck (a global view).

B. Definitions of fairness

What is a *fair* share of a resource? There are several well-known definitions of fairness, and we take the list below from the work of the TMRG.

In *max-min* fairness [12], each flow's throughput is at least as large as that of all other flows which have the same bottleneck. In this scheme each flow's demand is met, with the minimum demand (request for allocation) achieving the maximum allocation of resource. This assumes that the flow's demand is known, or (in the absence of this knowledge or no other resource usage model), that all flows effectively receive an equal share of the resource.

The goal of *proportional fairness* [13] is to maximise the utility function $U = \sum_1^N \log T_n$ for a given set of N flows, where T_n is the throughput of flow n. However, the implicit assumption that the utility can be modelled as a $\log$ function has not been justified.

Of course, *weighted* versions of max-min and proportional fairness are also possible, to reflect, for example, different assignments of capacity. With both max-min fairness, and proportional fairness, there is also the assumption that resource allocation can be controlled. In a best-effort IP network, we are typically unable to control resources on an end-to-end basis, but we may be able to *measure* usage of resources, especially (but not exclusively) at end-systems. In our aim to create a simple and practical methodology for assessing fairness, it would be beneficial to use a metric that is easily facilitated through some measurement related to a flow's performance, e.g. measurement of a flow's end-to-end throughput.

Whilst the metrics listed above focus on throughput, other proposals suggest using different measures to assess fairness. For example, there are proposals to use end-to-end delay for file transfers [14]. In [15], a strong case is made as to why throughput measurements should not be used for assessing fairness, but instead some notion of 'cost' should be considered. Motivated by [15], in [16], the case is reasserted for consideration of best-effort traffic to be considered. In keeping with [16], we choose to use end-to-end throughput, as it remains applicable to assessment of flow performance, is widely used, is easily understood and is straightforward to measure. However, our metric (Section II-D) is general and could also be applied using end-to-end delay or cost, if required.

C. Jain's Fairness Index

As mentioned above, Jain's Fairness Index (JFI) [9] is widely used for assessing *system-wide* fairness, as in

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

Equation (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.

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

An obvious approach to examining system-wide fairness over time is simply to evaluate the JFI at given instances during the period of interest, as in Equation (2), where t, in practice, is discrete. $r_n(t_j)$ is then 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, t was every second. So, J is the mean value of $J(t)$ values over a given time period.

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

The definition of JFI means that it may be difficult to determine the degree of relative unfairness between the flows. To illustrate, in Figure 1 we have created an artificial situation with two flows. Flow 2 is held constant at 100 and the value of Flow 1 is varied from 1 to 10000 (the units are immaterial). The plot shows the value of JFI (Equation 1) as the ratio Flow 1 / Flow 2 changes: the ratio has a range of four orders of magnitude, whilst the JFI has the effective range $[0.51, 1.00]$.

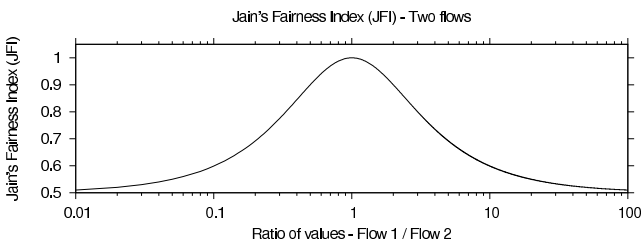


Figure 1. The range of JFI for two flows, Flow 2 = 100 (no units)

Also, we find that this effective range depends on the value of N. Furthermore, it is clear from Figure 1 that difference in say, 0.1, between JFI values has different significance, depending on the actual values of JFI being considered, i.e., the difference between $J = 0.9$ and $J = 0.8$ has a different significance in flow ratios than the difference between $J = 0.6$ and $J = 0.5$ So, it is not possible to use JFI easily in comparative analyses.

D. Resource usage and relative capability

An implicit assumption in JFI is that all of the processes being measured are equally capable of consuming the resource for which they are competing, and this is indeed the general assumption made in previous work [17], [18], including our own [6], [19]. However, when examining network flows, this is not necessarily true: some protocols may attain better performance than others given the same network conditions. It is thus necessary to take into account the flows' actual *capabilities*, in terms of the resources that it is *possible* for a flow to consume. That is, when making direct comparisons between resource consumption, JFI does not take into account the *relative capability* of the processes that are being evaluated, and so biases may result. So, we propose a different metric when considering fairness, one that is designed to be simple but allows:

- weights to be applied that reflect relative capability, given specific resource provisioning.
- comparative assessments to be made, based on relative capability.

Further, we choose to reflect the following characteristics in the output of our metric, in comparison to JFI:

- to be able to make comparisons of fairness on a *per-flow* basis, not just a *system-wide* basis.
- to enable an assessment of fairness over time (as well as a summary statistic), allowing observation of *per-flow* and *system-wide* dynamics.

A summary of the important definitions for our metric is given in Table I for convenience, and are defined in the remainder of this Section.

TABLE I.
SUMMARY OF IMPORTANT DEFINITIONS

Defn	Name	Eqn
$U_n(t)$	Normalised Resource Usage (NRU)	3
$R_n(t)$	Resource Share Ratio (RSR)	4
U_n	flow NRU (mean of the set $\{U_n(t)\}$)	5
U_N	mean system NRU (mean of the set $\{U_n\}$)	6
U_{N+}	fair system NRU	7
U_{N-}	unfair system NRU	8

E. Normalised Resource Usage (NRU)

In order to provide a richer view of the fairness information, we use a metric which is based on the ratio of resource usage of an individual flow with respect to its expected capability: the *Normalised Resource Usage (NRU)* [20]. The NRU metric, $U_n(t)$, for a flow n with throughput $r_n(t)$ at time t is defined in terms of the *Resource Share Ratio (RSR)*, $R_n(t)$:

$$U_n(t) = 10 \log_{10}(R_n(t)) \tag{3}$$
$$R_n(t) = w_n(t) r_n(t) \tag{4}$$

where w_n is a weight which reflects the relative capability of the flow under the conditions being examined. Key to this metric is the evaluation of $w_n(t)$, which we address in due course (Section II-G). The use of the $10 \log_{10}()$ deciBel term is for convenience of representing large and small values. When $U_n(t) = 0$, then flow n is receiving a fair share of the available resource. When $U_n(t) < 0$, then flow n is receiving less than its fair share of the resource. When $U_n(t) > 0$, flow n is receiving more than its fair share. This makes it easy to make relative comparisons of

fairness between flows: if any flows have $U(t) < 0$, then they are receiving less than their fair share of the resource, compared to their expected resource usage, regardless of the performance of other flows.

For a flow n with a set of values $U_n(t)$, over a given time-period, we can generate a summary, the *flow NRU*, by taking the mean, U_n, of the values in $U_n(t)$:

$$U_n = \overline{\{U_n(t)\}} \tag{5}$$

F. System-wide summaries

While time-based data sets let us view detailed dynamics, system-wide summaries are also important to allow comparative analyses to be made. We use our definition of $U_n(t)$ to generate summaries as follows.

Assuming a system has N flows, $1 \leq n \leq N$, a system-wide summary can be obtained by taking the mean, U_N, of U_n for all N flows. However, such a mean could hide unfairness, as positive and negative values of U_n would cancel out. So, we produce also U_{N+} and U_{N-}:

$$U_N = \overline{\{U_n\}} \tag{6}$$
$$U_{N+} = \overline{\{U_{n+}\}}, \{U_{n+}\} \subset \{U_n\} \forall U_n \geq 0 \tag{7}$$
$$U_{N-} = \overline{\{U_{n-}\}}, \{U_{n-}\} \subset \{U_n\} \forall U_n < 0 \tag{8}$$

where U_{N+} is the mean of the fair (zero) or better (positive) flow NRU values, and is called the *fair system NRU*; U_{N-} is the mean of all the unfair (negative) flow NRU values, and is called the *unfair system NRU*.

TABLE II.
INTERPRETATION OF SYSTEM NRU VALUES

	U_{N+}	U_{N-}	Case	Comment
unfairness	-	-ve	A	all flows unfair
	+ve	-ve	B	some flows fair or better, some flows unfair
	0	-ve	C	some flows fair, some flows unfair
fairness	+ve	-	D	all flows fair or better
	0	-	E	all flows fair

To explain the use of U_{N+} and U_{N-}, we refer to Table II. The combination of values ("$-$" denotes no value) can be grouped into those combinations that indicate a fair system and those that indicate unfairness. In the "Comment" column, 'fair' and 'unfair' are compared to the performance of the flows when the individual flows are run in a fair system. For Case A, there are no flows that have fair treatment, so the system as a whole is unfair. For Case B, some flows get more than their fair share and some less – so something in the system and/or the behaviour of some of the flows is causing unfairness to other flows. In Case C, no flows are getting more than their fair share, but some are getting less, so something in the system-wide behaviour or the behaviour of the flows themselves, is causing the unfairness. For the remaining two cases, all flows are either performing as well as in a fair system (Case E), or some are performing better than in the fair system (Case D).[3]

JFI cannot make this kind of distinction – it only indicates whether the system as a whole is fair. Additionally, use of the deciBel units allows us to make comparative performance analyses using familiar engineering semantics, which is not possible with JFI.

G. Weights for NRU

Practical calibration form an important distinguishing feature in our methodology and use of our new fairness metric: calibration tests are used to assign weights (Equation (4)) that reflect each flow's capability to consume the available resource. From Equation (4), we define:

$$w_n(t) = 1/_R r_n(t) \tag{9}$$
$$w_n = 1/_R r_n \tag{10}$$

where ${}_R r_n(t)$ is the expected throughput of that flow under the conditions being examined, and ${}_R r_n$ is $\overline{{}_R r_n(t)}$, the mean value of the set of expected throughput values. Equation (9) is a general expression, and we simplify this for our needs by using Equation (10), which represents the weight as a scaling factor, evaluated from the mean throughput values that we measure in our calibration tests (see Section IV). In our case, the value of w_n for each flow is taken from the appropriate column of the two values (Flows 1 and 2) in Table IV, at each RTT value. The construction of Table IV is explained in Section III-A.

III. METHODOLOGY AND METRICS

We have taken a practical approach, generating data flows using a modified version of the tool *iperf* [4], our modifications allowing easy use of of different TCP variants[5]. 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 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, with respect to these throughput measurements.

A. Testbed

Our testbed[6] set-up was the well-known dumbbell arrangement as depicted in Figure 2 and used in previous similar studies [7], [8], [19], albeit at higher speeds (100s of Mb/s to nearly ∼1Gb/s). This simple testbed helps to reduce the factors of error or unknown behaviour that

[3]Case D is included for the sake of completeness, but in practical situations, it may not occur.

[4]`http://dast.nlanr.net/Projects/Iperf/`

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

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

may affect the results and concentrate on the protocol behaviour. As noted in [21], "Simple topologies, like a "dumbbell" topology with one congested link, are sufficient to study many traffic properties."

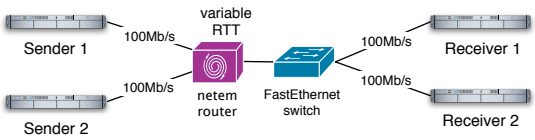


Figure 2. Testbed configuration

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, 100ms, 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[7], rather than tuning the stack for high-speed operation (as in [7], [8], [19]), 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 fragmentation): all these being default settings.

The 'netem router' ran Linux kernel version 2.6.18 and the package *netem*[8] 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 ∼1ms). We then generated single TCP flows and observed that all TCP variants and DCCP/CCID2 reached peaks of ∼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 3(a), 4(a), 5(a), 6(a) and 7(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 3(b), 4(b), 5(b), 6(b) and 7(b).

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

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

B. Weights

We noted earlier that to evaluate the NRU we require weights, as in Equation 10. The weights represent expected throughput values, which are evaluated from the throughput measurements detailed in Section IV-F. These are taken directly from Table IV. For example, when DCCP/CCID2 is Flow 1 and Compound is Flow 2, at RTT=25ms, we use the values of 43Mb/s and 51Mb/s, respectively, for ${}_Rr_n$ for DCCP/CCID2 and TCP Compound.

IV. Protocol behaviour

In this section, we describe briefly the behaviour of the individual 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.

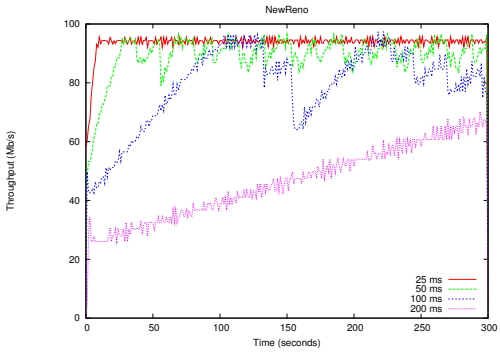
Our selection of protocols is somewhat arbitrary: we have chosen protocols that are widely used (or are likely to be widely used, in the case of DCCP/CCID2), are readily available for use within our experimental platform, and are of interest to a user community as well as the research community. Also, we have chosen to test against DCCP/CCID2 from [22], rather than against TCP NewReno, as there is already a body of work, including our own, comparing against TCP NewReno [6]–[8], [19]

A. TCP NewReno

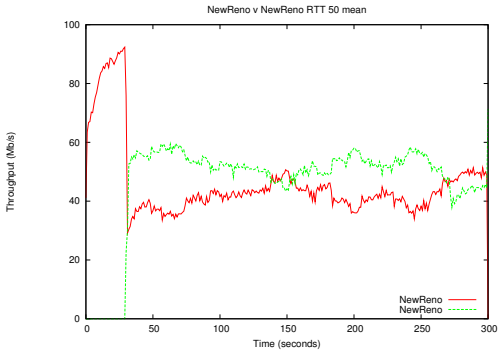
The basic congestion control algorithm in TCP NewReno is well known [1]. To control transmission, a *congestion window (cwnd)*, is subject to Additive Increase Multiplicative Decrease (AIMD) behaviour:

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

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 3(a), we note that TCP takes longer to achieve higher throughput as the RTT increases.

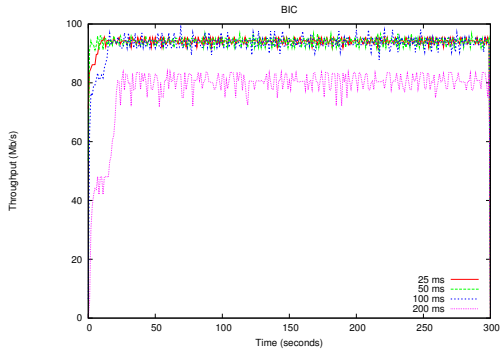


(a) Single flow, at various RTT

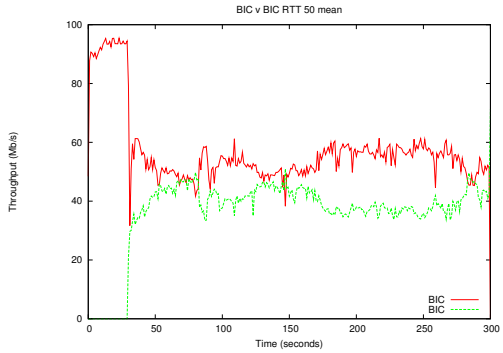


(b) Two flows, RTT=50ms

Figure 3. NewReno: typical behaviour on our testbed



(a) Single flow, at various RTT



(b) Two flows, RTT=50ms

Figure 4. BIC: typical behaviour on our testbed

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 4(a), compared to TCP (Figure 3(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}.

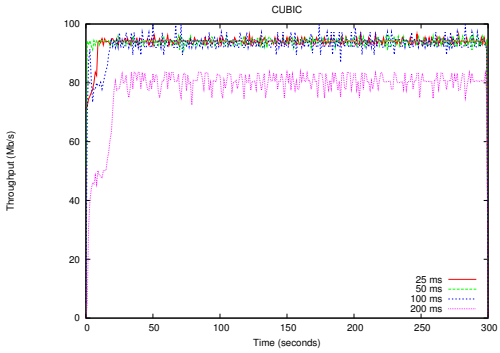
In Figure 5(a), compared to TCP (Figure 3(a)), CUBIC achieves higher throughput and more quickly at larger RTT values.

D. Compound TCP

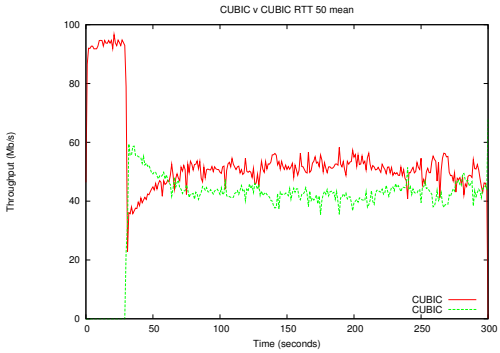
Compound TCP [4] is designed to adapt its behaviour by use of a scalable delay-based component. The main objective of Compound TCP is to be friendly to TCP NewReno, but to increase throughput more quickly in the congestion avoidance phase. A delay-based component, $dwnd$, and $cwnd$ are used together with the advertised window from the receiver, $awnd$, to determine the sending rate of a Compound TCP flow. The number of backlogged packets are estimated using RTT measurements from successfully acknowledged packets, and used with a threshold value, γ, to evaluate a final value for $dwnd$. The TCP sending window, wnd, becomes:

$$\begin{aligned} OnAck : cwnd &= cwnd + 1/(cwnd + awnd) \\ wnd &= \min(cwnd + dwnd, awnd) \end{aligned}$$

The evaluation of $dwnd$ is as follows:

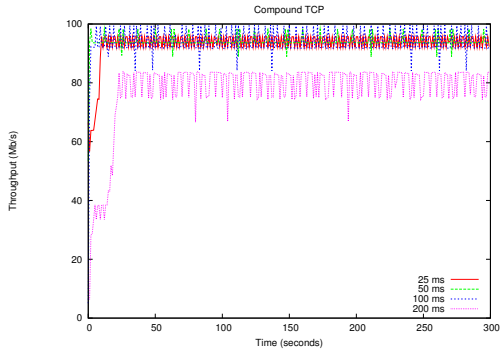


(a) Single flow, at various RTT

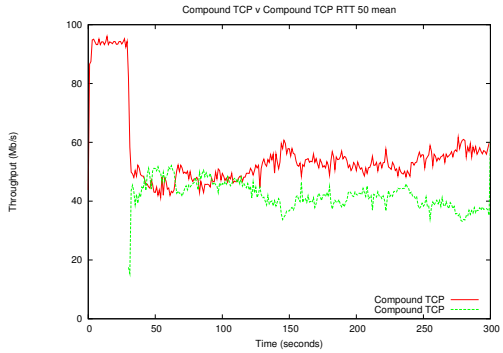


(b) Two flows, RTT=50ms

Figure 5. CUBIC: typical behaviour on our testbed



(a) Single flow, at various RTT



(b) Two flows, RTT=50ms

Figure 6. Compound: typical behaviour on testbed

$$
\begin{aligned}
OnACK : dwnd &\leftarrow dwnd + \alpha.dwnd^k,\ D < \gamma \\
dwnd &\leftarrow dwnd - \eta.D,\ D \geq \gamma \\
OnLoss : dwnd &\leftarrow dwnd.(1-\beta)
\end{aligned}
$$

where k, α, β and η allow the protocol to be tuned: the choices made currently are $k = 0.75$, $\alpha = 0.125$, $\beta = 0.5$ and $\eta = 1$. D is the difference between the smoothed RTT backlogged packet calculation and the non-smoothed RTT backlogged packet calculation for the flow. γ is evaluated dynamically as a function of $cwnd$ and RTT, and is constrained to the range $5 \leq \gamma \leq 30$.

Compound TCP is implemented in Windows Vista, Windows Server 2008, and available as a hotfix to Windows 2003 server and Windows XP 64-bit[9]. Compound TCP is also available for Linux. The implementation of Compound TCP used in our study is Caltech's Linux patch[10], which is written to the same specification used for the Windows implementation. As NewReno, BIC and CUBIC are all available in Linux, the Caltech implementation lends itself for easy use within our testbed, and we do not have to factor into our analysis any differences due to the behaviour of operating system if running Compound TCP under Windows.

In Figure 6(a), compared to TCP (Figure 3(a)), Compound achieves higher throughput and more quickly at larger RTT values.

[9] `http://support.microsoft.com/kb/949316`
[10] `http://netlab.caltech.edu/lachlan/ctcp/`

E. DCCP/CCID2

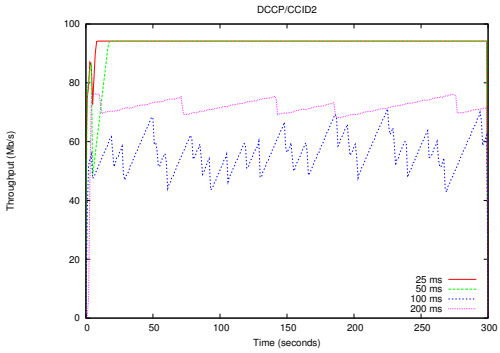
The Datagram Congestion Control Protocol (DCCP), "... is a transport protocol that provides bidirectional unicast connections of congestion-controlled unreliable datagrams. DCCP is suitable for applications that transfer fairly large amounts of data and that can benefit from control over the tradeoff between timeliness and reliability." [5]. 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) [10] is defined to be "TCP-like congestion control", i.e. as close as possible to the AIMD behaviour described in Section V-B[11]. 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 7(a), we note that although the pattern of throughout is similar to NewReno (Figure 3(a)), DCCP/CCID2 does not behave exactly the same, again to be expected from its design (as will be explained in our analysis in Section V-B).

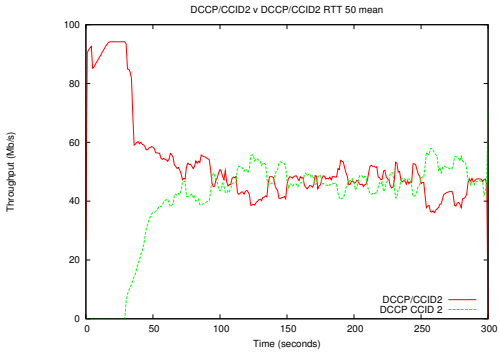
F. Throughput

When two flows of the same protocol are transmitted across the same path at the same time, we see that they

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



(a) Single flow, at various RTT



(b) Two flows, RTT=50ms

Figure 7. DCCP/CCID2: typical behaviour on testbed

achieve fairness. In Figures 3(b), 4(b), 5(b), 6(b) and 7(b) we see examples of 2 flows and can see that for each respective protocol, there appears to be good fairness. Table III shows the mean value of JFI from Equation (1), 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 IV. It is to be noted that, beyond 100ms (at higher BDP), Compound has the highest throughput values for both flows, and so has the best network utilisation overall.

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

	RTT (ms) [BDP (MB)]				Example
	25 [0.31]	50 [0.62]	100 [1.25]	200 [2.50]	Figure Thr'put
NewReno	0.93	0.94	0.93	0.93	3(b)
BIC	0.91	0.91	0.89	0.92	4(b)
CUBIC	0.90	0.90	0.90	0.90	5(b)
Compound TCP	0.97	0.95	0.93	0.90	6(b)
CCID2	0.96	0.95	0.97	0.91	7(b)

It should be noted that, when the order of flows is reversed, there are slightly different results (see Appendix A), but the overall trends are still correct and the conclusions drawn in this paper still stand.

V. INTER-PROTOCOL BEHAVIOUR

We now look at interaction of our chosen protocols by examining the behaviour of two flows across the testbed.

TABLE IV.
THROUGHPUT VALUES (MB/S) 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	39	55	40	54	38	56	38	56
BIC	33	61	33	61	32	63	34	60
CUBIC	33	61	33	62	33	61	33	62
Compound TCP	43	51	52	42	49	43	44	43
CCID2	43	51	47	47	45	48	33	32

Flow 1 was started at 0s and was a DCCP/CCID2 flow. Flow 2 was started at 30s and was one of the TCP variants: NewReno, BIC, CUBIC or Compound TCP. We executed five runs of each pair-wise experiment for each of the RTT values as explained in Section III-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). This means that for each pair-wise combination there are 1200 throughput measurements used per RTT value in our evaluation.

A. Summary of observed behaviour

The throughput values were used with Equation (1), to assess a summary of system behaviour. We also show for each pair-wise experiment, with the tuple $\langle U_{N-}, U_N, U_{N+} \rangle$ how the trends in fairness vary with RTT. We use U_N, plotted for each RTT, and U_{N+} and U_{N-} plotted in a similar fashion to error-bars to show the spread of the flow NRU values at that RTT. We connect with a line the values of U_N at the various RTTs in order to illustrate the trend in the NRU values. As we have only two flows, the values of U_{N+} and U_{N-} are, respectively, the mean flow NRU values for each of the two flows. As in all cases the TCP variant has higher throughput than DCCP/CCID2, U_{N+} is always for the TCP variant and U_{N-} is for the DCCP/CCID2 flow.

As a summary of the behaviour, the values of JFI generated using Equation (1) are given in Table V (the mean JFI values and standard deviations over five runs) and shown in Figure 8. Table VI 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, CUBIC and Compound 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, CUBIC and Compound 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 IV), it can be seen that DCCP/CCID2 becomes increasingly poor at utilising the available capacity (Figure 7(a)), whilst BIC (Figure 4(a)),

CUBIC (Figure 5(a)) and Compound (Figure 6(a)), have better performance at higher BDP, as per their design. So, the unfairness observed in Figure 8 may not necessarily be a concern. Note, however, that as the RTT increases beyond ~100ms, the fairness improves a little as DCCP/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 the TCP variants, though we have not yet examined this explicitly (also see the graphs in Appendix A).

B. DCCP/CCID2 vs NewReno

As might be expected, there is good fairness between DCCP/CCID2 and NewReno, with $J > 0.90$, as seen in Table V and visually in Figure 8. Various differences in the CCID2 behaviour, however, mean that it will not perform as well as NewReno. This is clearly visible in Figure 9: we see that the U_N, U_{N+} and U_{N-} values give a pretty flat trend. In Table VI, 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 100ms). Section 3.1 of [10] summarises the similarities between SACK-based TCP (as NewReno is) and DCCP/CCID2:

- 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 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.

So, overall, whilst we would expect to see good fairness between TCP NewReno and DCCP/CCID2, the differences in design listed above are visible in the NRU analysis in Figure 9.

C. DCCP/CCID2 vs BIC, CUBIC & Compound

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

In the case of both BIC, CUBIC and Compound run against DCCP/CCID2, we see from Figure 8 and Table V 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 or CUBIC, and

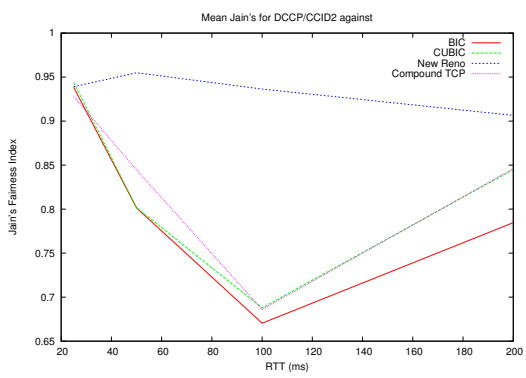


Figure 8. Pair-wise behaviour, DCCP/CCID2 flow first (See Table V)

TABLE V.
⟨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 8)

	RTT (ms) [BDP (MB)]			
	25 [0.31]	50 [0.62]	100 [1.25]	200 [2.50]
Reno	0.94, 0.01	0.96, 0.01	0.94, 0.02	0.91, 0.03
BIC	0.94, 0.02	0.80, 0.07	0.67, 0.11	0.78, 0.1
CUBIC	0.94, 0.02	0.80, 0.08	0.69, 0.11	0.85, 0.1
Compound TCP	0.93, 0.02	0.85, 0.05	0.69, 0.11	0.85, 0.9

TABLE VI.
THROUGHPUT (MB/S) FOR PAIR-WISE TESTS AGAINST DCCP/CCID2 (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	56	38	52	42	41	50	32	36
BIC	40	54	26	67	16	74	23	70
CUBIC	40	54	25	68	16	74	27	67
Compound TCP	38	56	30	63	16	74	28	65

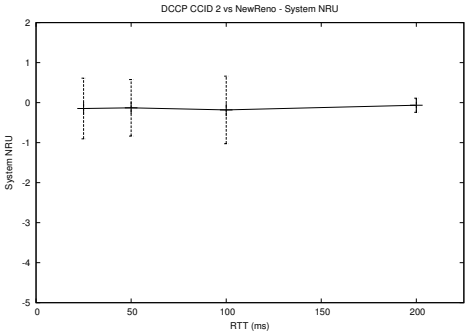


Figure 9. DCCP/CCID2 vs TCP NewReno - NRU

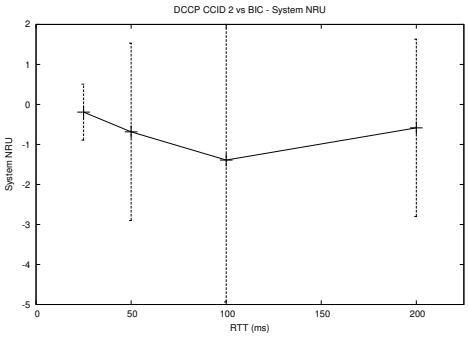


Figure 10. DCCP/CCID2 vs BIC - NRU

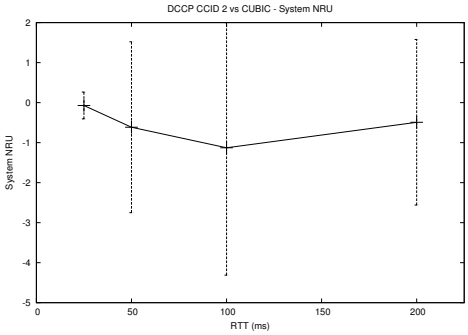


Figure 11. DCCP/CCID2 vs CUBIC - NRU

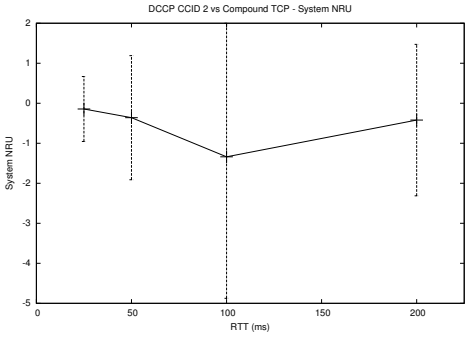


Figure 12. DCCP/CCID2 vs Compound TCP - NRU

DCCP/CCID2. In Table VI, we see that BIC and CUBIC always have 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 the high-speed variants are designed for high BDP paths, whereas DCCP/CCID2 is designed to exhibit "TCP-like" behaviour. So, for all the protocols, we must take into account their respective 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 the TCP variant. *Is the higher throughput of the TCP variant 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 VI (which records the mean throughputs of the *pair-wise* experiments) and Table IV (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.

Note that the definition of NRU in Equations 3, 4, and the use of the weights as described in Section II-G is such that the NRU explicitly includes this normalisation of the throughput.

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 IV for CCID2, and the values at the same RTT in Table VI. We find that at RTT=25ms, the throughputs of both DCCP/CCID2 and BIC are similar: DCCP/CCID2 (Table IV Flow 1) is 43Mb/s, and BIC (Table IV Flow 2) is 51Mb/s; DCCP/CCID2 (Table VI Flow 1) is 40Mb/s, and BIC (Table VI Flow 2) is 54Mb/s. So, there is very good fairness, arguably even better than the value of JFI=0.93 suggests (from Table V).

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 IV Flow 1) is 45Mb/s, and BIC (Table IV Flow 2) is 49Mb/s; DCCP/CCID2 (Table VI Flow 1) is 15Mb/s, and BIC (Table VI 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.

We observe this clearly in the NRU plots for BIC (Figure 10), CUBIC (Figure 11), and Compound BIC (Figure 12). We see the great difference in value between U_{N+} and U_{N-}, and observe the general trend for U_N. We note that greatest unfairness in all three cases is at 100ms, and that fairness improves at 200ms.

TABLE VII.
SYSTEM NRU TUPLES FOR 14 FLOW TEST IN NS2 (SEE FIGURE 13(A))

RTT (ms)			
25	50	75	100
$\langle -, -10.51, 3.21 \rangle$	$\langle -, -11.70, 1.94 \rangle$	$\langle -, -12.28, 1.80 \rangle$	$\langle -, -12.97, 1.68 \rangle$
RTT (ms)			
125	150	175	200
$\langle 0.02, -15.57, 2.56 \rangle$	$\langle 0.10, -16.98, 1.57 \rangle$	$\langle 0.10, -17.74, 2.88 \rangle$	$\langle 0.10, -18.47, 2.44 \rangle$

TABLE VIII.
JFI VALUES FOR 14 FLOWS (SINGLE RUN) (SEE FIGURE 13(B))

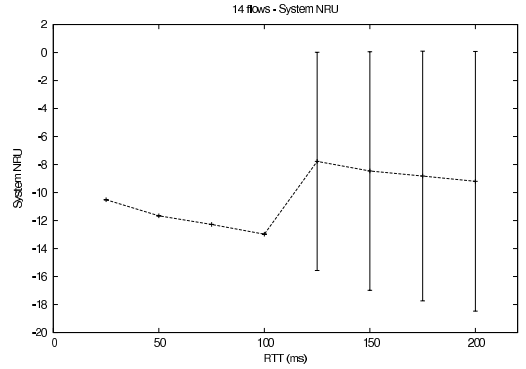
	RTT (ms)							
	25	50	75	100	125	150	175	200
JFI	0.51	0.30	0.25	0.24	0.22	0.20	0.20	0.19

D. Larger numbers of flows

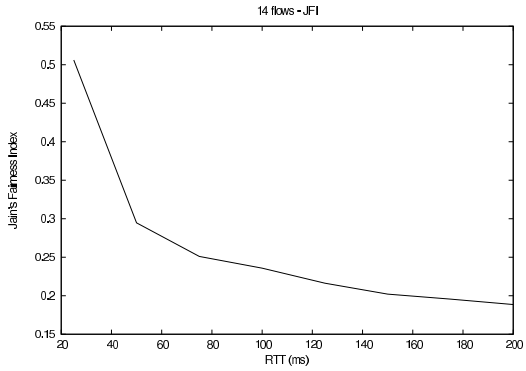
We now show the use of the NRU with larger numbers of flows using simulation. Note that these simulation results should not be considered as rigourous as those for our pair-wise tests above: our focus is to show the characteristics of the NRU and the actual protocol behaviour is not important.

We simulated a 14-flow system using *ns2* patched with the *ns2* Linux extension from [23], allowing Linux kernel code for different transport protocols to be executed from within *ns2* . We used 14 different TCP variants, one of each type supported by [23]. We used the same scenario and topology as for our testbed albeit with 14 senders and 14 receivers. Flows were started at 30s intervals (from $t = 0$) until all flows were active. The simulation lasted for 900s and we analysed data points from $t = 420s$ to $t = 900s$ in order to avoid any start-up artefacts from the flows. Table VII, Table VIII and Figure 13 shows the JFI and system NRU values (plotted as the tuple $\langle U_{N+}, U_N, U_{N-} \rangle$) for the 14 variants. The NRU weights were calculated in a similar fashion to that for our testbed, i.e. using a run of 14 flows of the same type. We conducted only a single run of the simulation in each case.

When we compare the JFI graph and NRU graph in Figure 13, we see that the JFI shows the system becoming increasingly unfair as the RTT increases, but the NRU shows that the mean system fairness improves after 100ms, as we take into account the relative capability of each protocol flow at those higher BDPs. Also, we see in the NRU plot that there are some flows that do perform better at 125ms and beyond, which is not visible in the JFI plot. Also, JFI depicts unfairness but masks the relative magnitude of reduction in performance, an aspect which is clearly visible in the NRU values (Table VII).



(a) System NRU (See Table VII)



(b) JFI (See Table VIII)

Figure 13. Comparing NRU and JFI for 14-flow inter-protocol behaviour

VI. 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, CUBIC and Compound. We have observed that, on our testbed,s with link speeds of 100Mb/s and at RTTs of 25ms, 50ms, 100ms and 200ms, DCCP/CCID2 has good fairness with TCP NewReno at all RTT values, though NewReno does achieve a slightly higher throughput. BIC, CUBIC, and Compound always achieved higher throughputs in our tests, becoming unfair above RTT=25ms, with greatest unfairness at 100ms, 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 and/or Compound, 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.

The use of our new metric, Normalised Resource Usage (NRU), allows us to have an appropriate assessment of fairness, weighted by the relative capability of each protocol within the environment being considered.

A. Limitations

The NRU's key benefit, the use of weights that provide the assessment of relative capability, is potentially also a limitation of the method. The key question is "How do we find appropriate values for the weights?" Our approach, with two flows (on our testbed) and 14 flows (in simulation), has used calibration runs in order to assess what each flow would achieve if competing with itself. Unfortunately, the use of calibration runs does not scale well as the number of flows increases. However, it should be possible to use other techniques to estimate the weights as the need arises, and the exact nature of the approximation is likely to be dependent on the system set-up. Meanwhile, it is to be noted that most studies in the literature typically make experimental assessments with small numbers of flows, e.g. two flows are used, as we have done in this paper, and also in past work, such as [7], [8]. So, for such purposes, our methodology is quite suitable and provides more accurate evaluation of fairness than Jain's Fairness Index (JFI).

B. Thoughts for future work

The "dip" in fairness as observed in Figure 8 (and the effects observed in Appendix A) need 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 the TCP variants in Figure 8 but not in Figure 14 (see Appendix A). 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.

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

Acknowledgements

We are grateful to the anonymous reviewers whose comments have helped us improve the quality of the presentation in this paper.

References

[1] V. Jacobson and M. J. Karels, "Congestion Avoidance and Control," *Proc SIGCOMM 1988*, pp. 314–329, September 1988. [Online]. Available: citeseer.ist.psu.edu/article/jacobson88congestion.html

[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 (PFLDNet05)*, Feb 2005.

[4] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM 2006*, Barcelona, April 2006.

[5] 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

[6] 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.

[7] 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.

[8] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental Evaluation of TCP Protocols for High-Speed Networks," *IEEE/ACM Transactions on Networking*, to appear. [Online]. Available: http://www.hamilton.ie/net/eval/ToNfinal.pdf

[9] 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.

[10] 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

[11] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," RFC 5166 (Informational), Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5166.txt

[12] J. Jaffe, "Bottleneck Flow Control," *IEEE Transactions on Communications*, vol. 29, no. 7, pp. 954–962, July 1981.

[13] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," in *Journal of the Operational Research Society*, vol. 49, 1998, pp. 237–252. [Online]. Available: http://www.statslab.cam.ac.uk/~frank/rate.pdf

[14] 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.

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

[16] S. Floyd and M. Allman, "Comments on the Usefulness of Simple Best-Effort Traffic," RFC 5290 (Informational), July 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5290.txt

[17] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental Evaluation of TCP Protocols for High-Speed Networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 5,

pp. 1109–1122, Oct 2007. [Online]. Available: http://www.hamilton.ie/net/eval/ToNfinal.pdf

[18] 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.

[19] 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.

[20] S. Bhatti, M. Bateman, D. Rehunathan, T. Henderson, G. Bigwood, and D. Miras, "Revisiting inter-flow fairness," in *Proceedings of BROADNETS 2008*, September 2008, pp. 585–592.

[21] S. Floyd and E. Kohler, "Internet research needs better models," in *Proceedings of HotNets–I*, October 2002.

[22] S. Bhatti, M. Bateman, and D. Miras, "A Comparative Performance Evaluation of DCCP," in *Proceedings of SPECTS 2008*, June 2008.

[23] D. X. Wei and P. Cao, "NS-2 TCP-Linux: an NS-2 TCP implementation with congestion control algorithms from Linux," in *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*. New York, NY, USA: ACM Press, 2006, p. 9.

Saleem Bhatti is a Professor in the School of Computer Science, University of St Andrews, Scotland, UK. His research interests include network architecture, protocols and systems; network and systems management; security, QoS and performance of networked systems; and issues related to the control-plane of networked systems. He holds a B.Eng (Hons), a MSc (Distinction), and a PhD all from University College London (UCL), UK.
`http://www.cs.st-andrews.ac.uk/~saleem/`

Martin Bateman is a Research Fellow in the School of Computer Science, University of St Andrews, Scotland, UK. His research include interests network QoS, performance of networked systems, video analysis and middleware systems. He holds a BSc (Hons) and MSc (Distinction) from Lancaster University, UK and a PhD from the University of St Andrews, UK.
`http://www.cs.st-andrews.ac.uk/~mb/`

APPENDIX A

Here, we present a short discussion, using the same testbed as the main paper, and the same experimental method (including number of runs, calibration runs, etc.). The purpose of this Appendix is to show that results could differ if were to start the DCCP/CCID2 flows as Flow 2 rather than as Flow 1, but that we have in the main paper captured what we consider as a worst case scenario.

If we run the experiments as before, but with the TCP variant as Flow 1, and DCCP/CCID2 as Flow 2, we get results for JFI values as shown in Table IX and Figure 14.

We note that the behaviour for BIC and CUBIC is quite different from that in Figure 8. The NRU plots are also differnt. The NRU plots should be compared as follows: Figure 9 with Figure 15; Figure 10 with Figure 16; Figure 11 with Figure 17; and Figure 12 with Figure 18.

There is still unfairness with DCCP/CCID2 but the effect is not the same for BIC and CUBIC. We observe that there is more distinction between BIC, CUBIC and Compound. Also, that the greatest unfairness is at 200ms and not 100ms: the "dip" at 100ms has disappeared for BIC and CUBIC, compared to when the DCCP/CCID2 flow was started first.

This behaviour needs to be investigated further in future work. We see in Figure 8, Table V and Table VI, that when the DCCP/CCID2 flow is Flow 1, that unfairness starts after 25ms. So, in our main text, we have effectively analysed a worse case compared to Figure 14.

TABLE IX.
⟨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 14)

	RTT (ms) [BDP (MB)]			
	25 [0.31]	50 [0.62]	100 [1.25]	200 [2.5]
Reno	0.89, 0.01	0.88, 0.1	0.90, 0.02	0.80, 0.05
BIC	0.92, 0.01	0.93, 0.01	0.85, 0.4	0.53, 0.19
CUBIC	0.93, 0.01	0.93, 0.01	0.83, 0.05	0.60, 0.14
Compound TCP	0.95, 0.01	0.83, 0.06	0.66, 0.13	0.86, 0.11

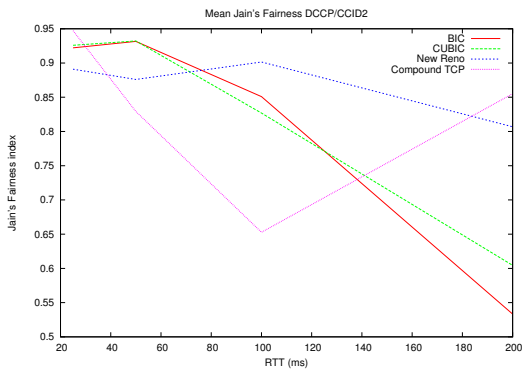


Figure 14. Pair-wise behaviour, TCP variant flow first (See Table IX)

TABLE X.
THROUGHPUT (MB/S) FOR PAIR-WISE TESTS AGAINST DCCP/CCID2, TCP VARIANT FLOW FIRST (MEAN OVER 5 RUNS, MEASUREMENTS FROM $t = 60s$ TO $t = 300s$)

	RTT (ms)							
	25		50		100		200	
Flow	1	2	1	2	1	2	1	2
NewReno	17	29	18	32	21	31	15	37
BIC	19	28	25	27	32	16	36	8
CUBIC	19	29	25	27	34	15	35	8
Compound TCP	52	42	65	28	66	16	66	28

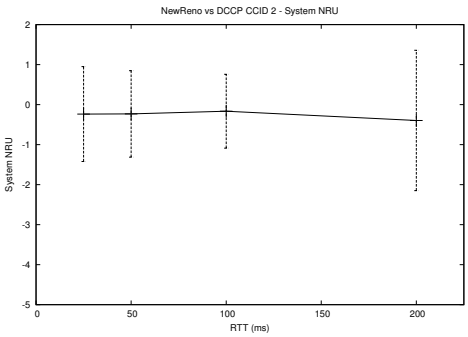


Figure 15. TCP NewReno vs DCCP/CCID2 - NRU

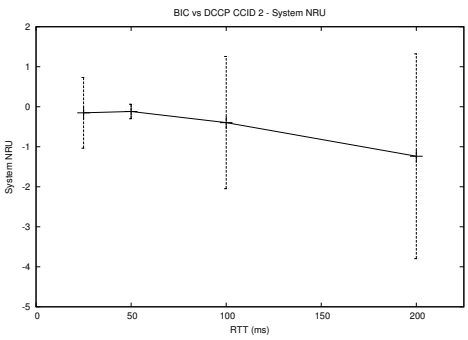


Figure 16. BIC vs DCCP/CCID2 - NRU

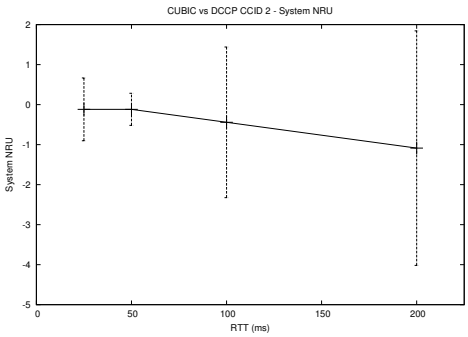


Figure 17. CUBIC vs DCCP/CCID2 - NRU

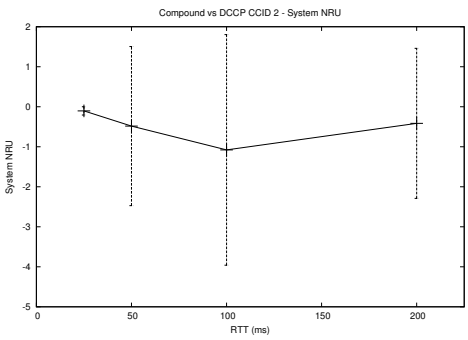


Figure 18. Compound TCP vs DCCP/CCID2 - NRU