# TCP testing: How well does *ns2* match reality?

Martin Bateman
CEPS
University of Central Lancashire
Preston, UK
mbateman@uclan.ac.uk

Saleem Bhatti
School of Computer Science
University of St Andrews
St Andrews, UK
saleem@cs.st-andrews.ac.uk

***Abstract*—New transport protocols continue to appear as alternatives to the Transmission Control Protocol (TCP). Many of these are are designed to address TCP's inefficiency in operating over paths with a high bandwidth-delay product (BDP). To test these new protocols, especially comparatively, and to understand their interactions, extensions to the *ns2* simulator allow real code from the linux kernel to be used within the *ns2* simulations. However, how does the performance of such configurations compare to test-bed experiments of the same configuration? Although, anecdotally, there are often comments within the research community about such issues, there are no studies that quantify the differences for a specific protocol suite. Using a simple testbed, we assess four different transport protocols in a comparative study to examine how well *ns2* matches reality. Our tests are all conducted at 100Mb/s over a wide range of delay and router buffer conditions: end-to-end delays from 25ms to 400ms, with end-to-end path buffering of 20% to 100% of the BDP. We find that in our simple configuration, there are significant differences in performance between *ns2* and the testbed.**

## I. Introduction

*ns2* [1] is widely used in the research community to evaluate protocols in a simulation environment. Although other simulators are widely available, *ns2* is free, the source code is available for inspection and modification, and there is a large user community that provides contributions in the form of extensions that allow new protocols and systems to be simulated. That same community has expressed concerns over the process of evolution of *ns2* , and the impact this might have had on how well *ns2* compares to use of protocols in the 'real world' [9]. However, *ns2* remains popular and development on *ns2* continues in parallel with the major development on *ns3*[2]. Meanwhile, work by Wei and Cao [21] allows real TCP code to be used in *ns2* simulations, so that more realistic simulations can be carried out.

It is important to understand in any specific case, just how well, or not, a simulation (any simulation, not just one based on *ns2* ) may relate to the real world scenario which it claims to simulate. Arguably, this is of particular importance when the scenario being simulated presents results for existing protocols (i.e. *post hoc*), rather than presenting results that concern, say, a new or experimental protocol that is still in development and is not yet widely deployed. This is because such results could affect deployment decisions and use of real world systems based on those results.

[1] http://nsnam.isi.edu/nsnam/
[2] http://www.nsnam.org/

While a complete and rigourous evaluation of *ns2* would be a large undertaking, our aim in this paper is to examine how well the behaviour of various flavours of TCP in *ns2* compare with a simple testbed set-up. The comparison is made by importing into *ns2* the same TCP code used in the testbed and running some tests on a simple network configuration. The evaluation involves a summary metric, *fairness*, based on measurements of end-to-end throughput for individual flows.

### A. Contribution and structure

We make two contributions in this paper:

- **Primary:** An original comparison of *ns2* against a testbed, for a specific network configuration, using TCP performance tests, which use the same Linux TCP code in the testbed and in *ns2* . We observe a significant difference in behaviour between *ns2* and our testbed.
- **Secondary:** An evaluation of throughput *fairness* between TCP flows operating on a testbed over a range of bandwidth-delay product (BDP) values and router path buffering (RB). Previous studies have not considered a range of RB sizes as we have, nor have they included Compound TCP.

After presenting our scenario for evaluation, and defining the *fairness* metric (Section II), we describe our configuration for the testbed and for *ns2* , with a description of our experiments (Section III). After considering our results with a discussion (Section IV), we conclude with some comments on possibilities for future work (Section V).

## II. Scenario

There is growing interest in more widespread use of TCP variants as parts of the user community begin to use more demanding applications (e.g., data-intensive applications). Users and especially administrators, however, are concerned with the impact these variants would have on the network. This is because they adopt different congestion control algorithms to normal (NewReno) TCP; algorithms that are potentially more 'aggressive' in their transmission behaviour. This is by design: the goal of these newer protocols is to make use of available network capacity that NewReno cannot utilise effectively due to its relatively conservative congestion control behaviour. The use of congestion control in TCP has been the key to the Internet's stability, so any change to this behaviour merits investigation. In order to investigate this

behaviour, a researcher may naturally turn to simulation with *ns2* in order to generate a set of experiments that are easily manageable, scalable, configurable and reproducible for their specific scenarios of interest, as reproducing and measuring those scenarios at scale in a testbed may not be feasible.

So, we evaluate the efficacy of the *ns2* simulations for a very specific set of cases: we show the interaction between TCP NewReno and four TCP variants[3] when operating over network paths with an end-to-end capacity of 100Mb/s. The flows being examined share a bottleneck link, where congestion occurs, invoking the congestion control behaviour of the respective TCP variant. That is, the TCP flow is operating over a network path with a range of bandwidth-delay product (BDP) values, and competing for resources at a bottleneck on that path.

Our direct experimental objective was to measure how the TCP variants compete with each other and how *fair* they are in their behaviour. We compared fairness by measuring the end-to-end throughput of the flow using a version of the well known tool, *iperf*[4], to generate flows and to report throughput values.

It is known that TCP behaviour can be effected by the end-to-end delay on the path (e.g., [17]) and by the buffering used on the path (e.g., [10], [18]). For the end-to-end path, we use a range of round-trip times (RTTs) – 25ms to 400ms, in 25ms steps – and a range of on-path buffering – 20% of the BDP to 100% of the BDP, in 20% steps.

We compare the results from *ns2* to those from a testbed using the same Linux TCP code in both cases. We use the *ns2* Linux enhancements from Wei and Cao [21] in order to import the TCP congestion control implementations from the Linux kernel version 2.6.22.6 into *ns2*. We configure a physical testbed and *ns2* with the same network topology and use the same Linux kernel for sending and receiving nodes. This means that in each of the two cases – simulation and testbed – the same code will be used for the control of the TCP flows generated. In this way, variations in implementation of the congestion control algorithm are removed and only the differences from the simulation and the testbed platform remain.

### A. Fairness

We evaluate fairness for both the simulated and testbed environments. Key to our evaluation is a definition of the *fairness* metric. We use the well-known Jain's Fairness Index (JFI) [13] to evaluate overall system fairness:

$$F(x) = \frac{(\sum x_n)^2}{N(\sum x_n^2)} \quad (1)$$

where, $0 \leq J \leq 1$, $N$ is the number of flows, and $x_n$ is the value of the resource attribute being assessed for flow $n$, e.g., $x_n$ is the measured end-to-end throughput. $J = 1$ means there is fairness across all flows; $J = 0$ indicates no fairness. In our experiments, we compare the pair-wise throughput of two flows, and evaluate fairness between a NewReno flow, and flow of each of the three other TCP variants in turn ($N = 2$). What is a *fair* share of a resource? There are several well-known definitions of fairness – a non-exhaustive discussion follows.

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* [14] 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, but we may be able to *measure* usage of resources, especially (but not exclusively) at end-systems.

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 [15], or some notion of 'cost' [4]. We choose to use end-to-end throughput, as it remains applicable to assessment of flow performance, is widely used, is easily understood, is straightforward to measure, and is sufficient for our comparative analysis.

In reality, for JFI with $N = 2$, as long as one of the flows has a non-zero value for $x_n$, we find $0.5 \leq J \leq 1$. There are other limitations of the JFI [3], but it is sufficient for our comparative analysis, and allows comparison with previous studies.

### B. TCP variants

We use an experimental testbed that attempts to control as many of the salient end-system factors as possible. The specific TCP variants to discuss in this paper were chosen using the following criteria:

- protocols which have attracted interest by the research community and will be of interest to the user community.
- protocols which have implementations that are readily available for deployment at the time of writing.
- protocols which are being used or being considered for use across the Internet as well in private IP networks.
- protocols which have source code available in Linux so that we can compare *ns2* and our testbed experiments.

We have chosen BIC [22], CUBIC [23] and Compound TCP [20]. These TCP variants have different designs, even though they are all designed to operate in environments with high

[3]Our complete results include data for all 14 variants of TCP available in Linux kernel version 2.6.22.6.

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

bandwidth-delay products. BIC and CUBIC use loss signals to trigger congestion control behaviour, whilst Compound TCP uses delay/RTT (round-trip-time) information. Each variant offers some improvement in end-to-end throughput compared to NewReno, in specific situations.

TCP NewReno is the variant of TCP that is the default in Windows XP, Windows Vista, and MacOS X (amongst others). BIC and CUBIC are the default versions of TCP in place of NewReno in Linux kernels over the past few years[5]. Compound TCP is available in Windows Vista, Windows Server 2008, and available as a hotfix to Windows 2003 server and Windows XP 64-bit[6]. Our implementation of Compound TCP is Caltech's Linux patch[7]: although we have not verified its relative performance compared to a native Windows implementation, Windows source code is not readily available to include into *ns2* , so we chose to use the Linux implementation.

Descriptions of the relevant behaviour of TCP variants used are summarised in the Appendix.

## III. Methodology

We have configured two experimental networks, one simulated and one testbed. We have used the default ("out-of-the-box") configuration for the network testbed and have configured *ns2* to be as similar as possible.

### A. Testbed

Our testbed set-up was the well-known dumbbell arrangement, as depicted in Figure III-A, and used in previous similar studies [11], [16], [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 may affect the results and concentrate on the protocol behaviour. As noted in [7], "Simple topologies, like a 'dumbbell' topology with one congested link, are sufficient to study many traffic properties."

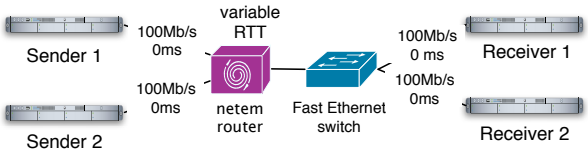



Fig. 1. Testbed configuration

The dumbbell configuration, consists of two senders, two receivers and a router to provide the network delay. All the nodes were Intel Xeon Dual Core 1.6Ghz with 2GB of DDR2 (667Mhz) memory and a PCI-X (64-bit, 133Mhz) bus. All network connections are 100Mb/s full-duplex 100Mb/s Ethernet. The senders and receivers are running Linux kernel version 2.6.22.6, while the router is running Linux kernel version 2.6.18. The package *netem* [8] was used at the router to introduce delay to the packet flows. On the *netem* router, the network delay is split equally between the forward and reverse paths. The two flows are sent from Sender 1 to Receiver 1 and from Sender 2 to Receiver 2, respectively.

Whilst there are various criticisms that can be levelled at this arrangement, our aim was not to show a production network scenario but to provide a set-up that would allow us to test the behaviour of the TCP variants when used on real hardware in a way that is easy to compare with *ns2* . Indeed, this very simple testbed helps us to reduce the factors of error or unknown behaviour that may affect the results. We are also aware of the work within the IRTF Transport Modelling Research Group[9], who are also looking to define mechanisms and metrics for evaluation of transport protocols, including fairness. The ongoing work within the TMRG suggests that a dumbbell topology is one scenario that can be used to evaluate transport protocol performance [6].

Our measurement runs consist of generating two flows, using *iperf* , for a pair-wise comparison: Flow 1 is from Sender 1 to Receiver 1, and Flow 2 is 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 60s to avoid initial synchronisation effects.

We use an *out-of-the-box* configuration for the end-systems (senders and receivers), rather than tuning the stack for high-speed operation, in order to gauge the performance under (arguably) the most likely configuration of the end-system. TCP Selective Acknowledgements (SACK, RFC2018) is enabled; the Window Scale Option, Protection Against Wrapped Sequence Numbers, and Round-Trip Time Measurement (RFC1323) were enabled; and MTU size is 1500 bytes (no IP fragmentation): all these being default settings.

| Parameter | Range | Increments |
|---|---|---|
| RB (% of BDP) | 20% – 100% | 20% |
| RTT | 25ms – 400ms | 25ms |

TABLE I
Experimental parameters for the testbed and *ns2*

### B. Testbed calibration and validation

The router buffer (RB) size (the on-path buffering) was set to a given percentage of the BDP at each RTT value, the BDP being evaluated as 100Mb/s multiplied by the RTT value. The RB and RTT values used are given in Table III-A. This means, for each TCP variant a total of 80 data points are produced.

Whilst router buffer sizing remains an open issue, it is clear that buffer sizing does affect TCP throughput [18], and so it would seem appropriate to consider it as a potential factor to test.

Of course, packet loss (due to congestion or transmission errors) can also affect TCP throughput [17], however, we leave that for further work.

[5]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.

[6]http://support.microsoft.com/kb/949316

[7]http://netlab.caltech.edu/lachlan/ctcp/

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

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

For each RTT value, the testbed is validated:

- by testing the RTT by the use of *ping* from sender to receiver: *ping* reported the delay was accurately configured (to within ∼1ms).
- by the use of *iperf* using UDP flows to ensure that a packet-level throughput of 100Mb/s was possible.

### C. Flow generation for the testbed

The TCP flows are generated using a modified version of *iperf* v2.0.2[10]: the modification allow us to switch easily between the TCP variants used on a per-flow basis.

Each experiment is run for 300 seconds. The first flow, Flow 1, is started at $t = 0s$ and the second flow, Flow 2, is started at $t = 60s$ to help avoid initial synchronisation effects. The calculation of JFI uses the throughput reported by *iperf* from 120s to 300s at 1s intervals, i.e. the first 119 measurements reported are ignored, to give the flows time to reach steady-state. Flow 1 is always TCP NewReno and Flow 2 is, in turn, TCP NewReno, BIC, CUBIC, TCP-Compound.

Each experiment run is repeated 5 times, with the mean of the 5 values taken as the measured value.

### D. ns2 set-up

We use *ns2* version 2.31 with the TCP Linux extension from Wei and Cao [21], running Linux Kernel version 2.6.22.6 (the same as the testbed sender and receiver nodes). *ns2* is set to simulate a dumbbell network topology shown in Figure 2. The bottleneck bandwidth of 100Mb/s is applied at the same time and on the same link as the variable RTT. We alter the router buffer size in line with the RTT and bandwidth to 20%, 40%, 60% and 100% of the BDP. The MTU was 1500 bytes and SACK is enabled.

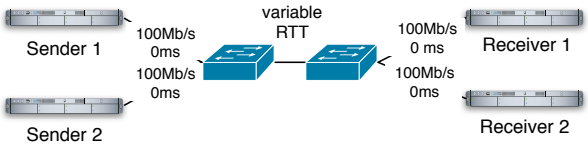



Fig. 2. Simulation configuration

## IV. Results

When we observe typical behaviour for the throughputs of the pair-wise flows, in a completely fair system, each flow would achieve a fair share (∼50Mb/s) of the bottleneck link, and the value if the JFI would be ∼1. However, this is not the case. Neither the testbed or the simulation result in fair flows. Furthermore, the behaviour observed on the testbed and in the simulation are different.

In total, we have 5 cases, as listed in Table II. The final column in the table references the figure which shows the difference in JFI values, between the simulation results and the testbed results. We use a subscript of $_S$ to denote the simulation results for the case, and a subscript of $_T$ to denote the testbed results for each case.

| Case | Flow 2 | JFI Figures | Difference surface Figures |
|---|---|---|---|
| *NR* | NewReno | 4(a), 4(b) | 5(a), 5(b) |
| *BI* | BIC | 4(c), 4(d) | 5(c), 5(d) |
| *CU* | CUBIC | 4(e), 4(f) | 5(e) 5(f) |
| *CO* | Compound TCP | 4(g), 4(h) | 5(g) 5(h) |

TABLE II
Cases for experiments (Flow 1 is always NewReno)

Also, it is instructive to consider how the values of the JFI change as the ratio of the throughput of the flows changes. In Figure 3, we show the value of JFI changes as the ratio of two fictitious flows, Flow 1 and Flow 2, changes. We note that as the value of the ratio Flow 1 / Flow 2 changes over 4 orders of magnitude, the value of the JFI has the range [0.51, 1.00] [3]: so a difference in JFI values of 0.6 could represent an order of magnitude difference in flow throughput.

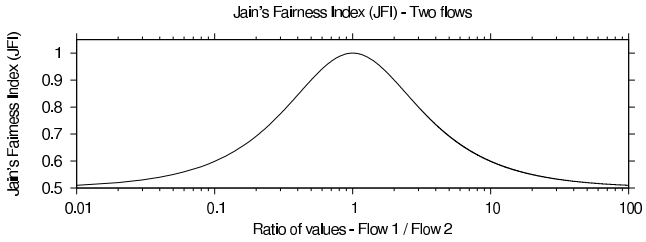



Fig. 3. The range of JFI values for two flows

### A. Comparison of simulation and testbed

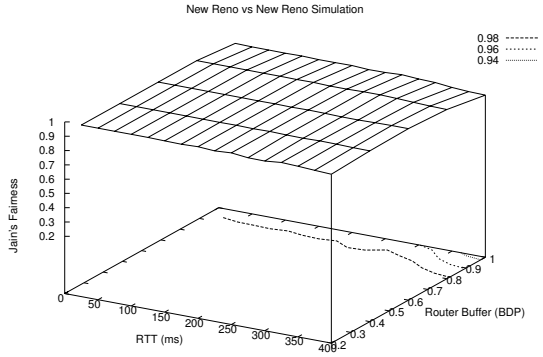
The discussion of our results here address our Primary contribution, showing that *ns2* and our testbed have poor agreement. To show in a single visualisation of the results as we vary the RTT and the path buffering, we use axis $\langle x, y, z \rangle$ as $\langle$ RTT, path buffer, JFI $\rangle$. For all cases, we can see the behaviour of 2 flows in Figure IV-A, where each subfigure shows the case for the simulation and the testbed.

Each plot for the simulation shows the JFI value for a given RTT at a given path buffer provision. Figures 4(a) and 4(b) show that TCP NewReno is fair to itself in simulation and testbed, which is to be expected.

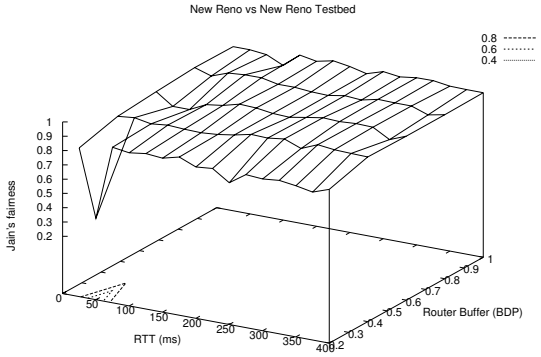
From Figure IV-A (left column), we see that *ns2* shows good fairness with TCP NewReno at all RB and RTT values for BIC and CUBIC, and very good agreement for Compound TCP, except at low values of RB. This is in contrast to the testbed (right column of Figure IV-A), which has very variable fairness for the TCP variants – each variant achieves higher throughput than TCP NewReno. The fairness decreases as the RTT increases (i.e. as the BDP increases), except when the RB values is large (when good fairness is observed).

Let us define $T$ as the matrix of values used to plot one of our graphs from our testbed. Each value in $T$ is the

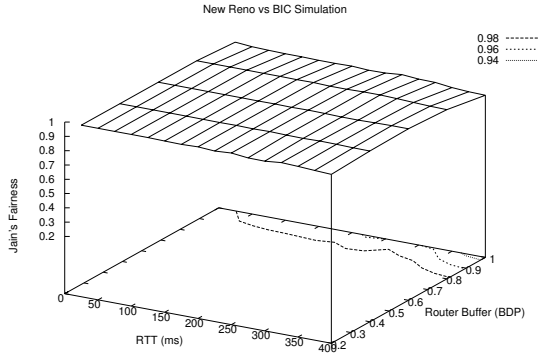
[10]The modified version of *iperf* is available, upon request, from the authors.



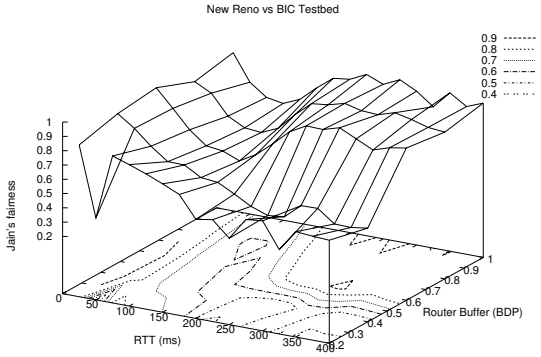

(a) Case $NR_S$ (simulation) (Min 0.93 Mean 0.99 Max 1.0)



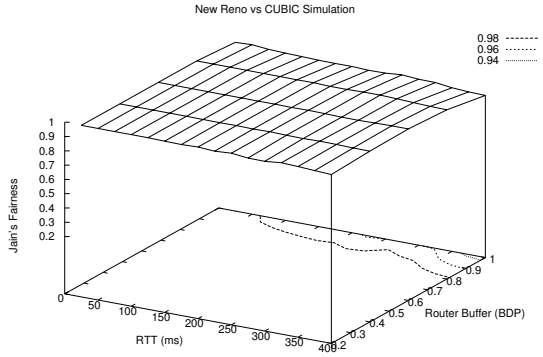

(b) Case $NR_T$ (testbed) (Min 0.40 Mean 0.92 Max 0.98)



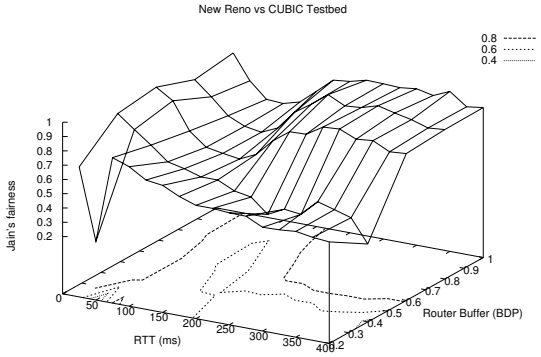

(c) Case $BI_S$ (simulation) (Min 0.94 Mean 0.99 Max 1.0)



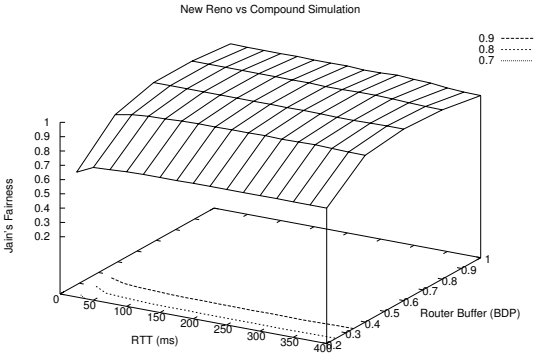

(d) Case $BI_T$ (testbed) (Min 0.37 Mean 0.71 Max 0.96)



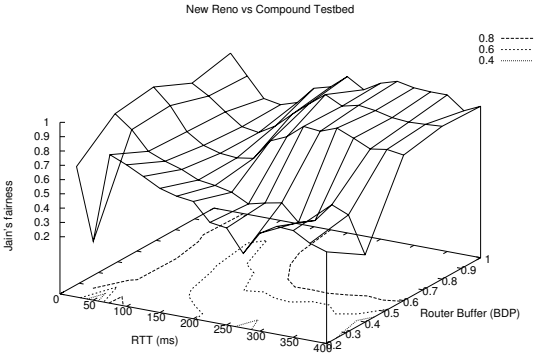

(e) Case $CU_S$ (simulation) (Min 0.93 Mean 0.99 Max 1.0)




(f) Case $CU_T$ (testbed) (Min 0.21 Mean 0.72 Max 0.97)




(g) Case $TC_S$ (simulation) (Min 0.67 Mean 0.93 Max 1.0)




(h) Case $TC_T$ (testbed) (Min 0.21 Mean 0.71 Max 0.99)

Fig. 4. Jain's Fairness Index (JFI) values for all cases

mean JFI value taken from the 5 JFI values generated by the 5 experimental runs for each combination of RTT and buffer size. Then, we define $T_\sigma$ as the matrix where each element is the standard deviation of the 5 experiment runs used to generate the corresponding value in $T$. Now, if $S$ is the matrix of JFI values from our simulation runs then we define $D = S - T$, as the difference in JFI values between simulation and testbed. If we now define $C_{95} = 2T_\sigma$, and $W = \text{abs}(D) - C_{95}$, then where the values in $W$ are zero or more we can say that we have 95% confidence that at the corresponding RB–RTT combination, the difference between the simulation results and the testbed results are statistically significant. We call the plot of $W$ for each case the *difference surface*. So, for clarity, in order to highlight the disagreement between the simulation and the testbed, in Figure IV-A, the left column plots the surface in a conventional form, whilst the right column uses a grey-scale, with darker colours showing greater difference.

As can be seen by comparing Figures 4(a) and 4(b), there is poor agreement between the simulation results and the testbed results for TCP NewReno. By examining Figures 5(a) and 5(b), we see that the differences are mostly (though not always) statistically significant. As we look at the cases for BIC, CUBIC and Compound TCP, we see that there is also significant disagreement between the simulation and testbed. The results for BIC and CUBIC seem consistent with other studies (e.g., at the lower RB values we can compare with [11], [16], [19], and at the higher RB values, with [1], [16]).

While there is disagreement between *ns2* and the testbed, we note that the situation improves as the RB and RTT values increase (towards the top right quadrant of the right column plots of Figure IV-A), and also at low RTT values (less than ∼75ms). However, we also note that in all cases, there seems to be poor agreement at low RTT and low RB (the bottom left corner of the plots in the right column of Figure IV-A). This is caused by what seems like it could be an outlier in the testbed, but it is present consistently in all experimental runs for all TCP variants so we believe this behaviour to be real.

We note that changes in router buffering (values of RB) can significantly affect the fairness (both increase and decrease fairness) for the same RTT values.

### B. Performance of TCP variants

The discussion of our results here address our Secondary contribution, examining the relative performance of the TCP variants we have tested.

It is to be noted that there is other work that looks at testing of TCP fairness (e.g., [5], [11], [16]), but none of those previous studies consider Compound TCP, or consider the range of path buffering that we have considered.

From Figure IV-A, we find that, overall, BIC, CUBIC and Compound TCP all show poor fairness to TCP NewReno. We would expect this at high BDP values (high RTT values), but we see that this is also true at low BDP values. Fairness improves when BDP is high and RB is also high, and seems worst at high RTTs and low RB as well as in the regions where RTT ∼200ms with RB above ∼0.4BDP.

Also, we note from the shape of the graphs in Figure IV-A that BIC, CUBIC and Compound TCP all show very similar behaviour towards TCP NewReno. So, whilst they may be unfair to TCP NewReno, they may be fair to each other, though this is an investigation for future work.

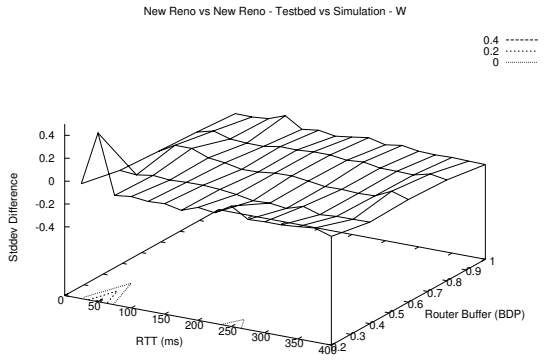
### C. Discussion and critical assessment

Previously, we have found that it is possible to get reasonable agreement between *ns2* and a testbed that is very similar to the one used in this paper [1]: trends of protocol behaviour were similar, and the JFI values were similar in magnitude, albeit the error-bars on the testbed results did not cover the results from the *ns2* simulations. In [1], the testbed and the *ns2* simulations were tuned for high-speed operation (1Gb/s paths), and the on-path buffering and end-system buffering were set to very high values (greater than the BDP). In this paper, however, we used *out-of-the-box* (default) configurations for both the test-bed and the *ns2* simulations. The relevant settings are given in Section III-D, showing that *ns2* and the testbed are configured as close to the same as is possible.

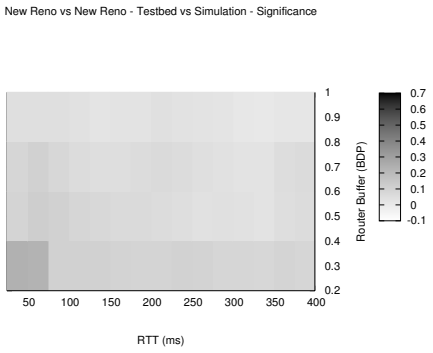
We have performed our analysis by using TCP NewReno always as Flow 1. It is quite possible that using the alternative TCP variant as Flow 1 will yield different results. Our previous experience with performing similar evaluations [2] shows that behaviour can indeed be different for NewReno and CUBIC against DCCP/CCID2 [8] on a testbed. However, in that evaluation such differences were visible only at low BDP, and were insignificant at high BDP (high RTT, above ∼100ms). So, if the experiments were repeated with Flow 1 as the TCP variant and Flow 2 as NewReno, we would expect the overall outcome to be similar – poor correspondence between *ns2* and the testbed.

Another factor to consider is response to loss. Lossy paths may be due to high-levels of congestion, or due to packets traversing links that suffer high-levels of bit-errors, which are not corrected before IP-layer or TCP-layer processing, so IP or TCP checksum errors result in packets being discarded. TCP is known not to respond well to lost segments, and verifying loss behaviour and channels with errors in *ns2* would also be useful.

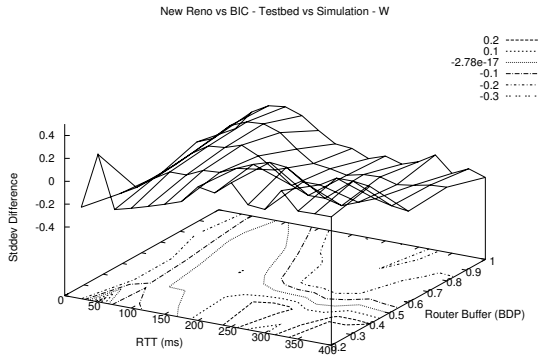
Jain's Fairness Index (JFI) has been utilised as it is widely used and understood, and allows us to make comparisons with other studies, if we so wish. Note that in considering fairness, JFI assumes that each process being evaluated (in this case, each TCP variant), is equally capable of consuming the resource. That is, for example, if TCP NewReno was not competing with any other flows, it would achieve the same throughput as any of the other TCP variants. However, this assumption does not hold, as the design goals of the other variants, and previous studies involving TCP NewReno, BIC and CUBIC, show that this is not true. So, in fact, Jain's Fairness Index may be unfair when assessing fairness as it does not take account of the relative capability of each process



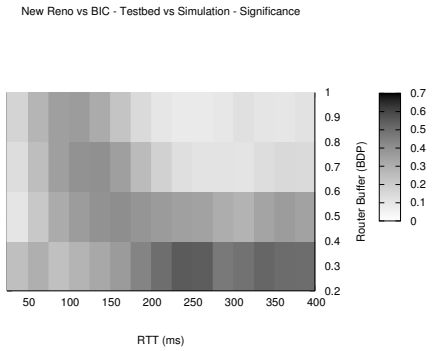

(a) Case *NR* - NewReno



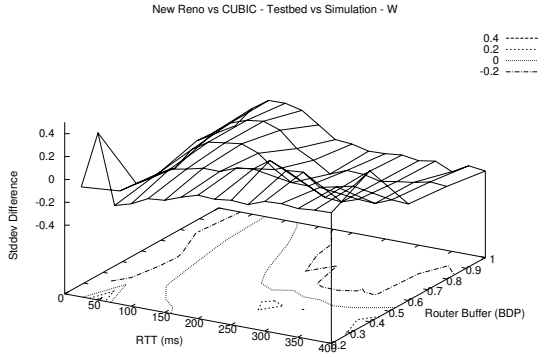

(b) Case *NR* - NewReno



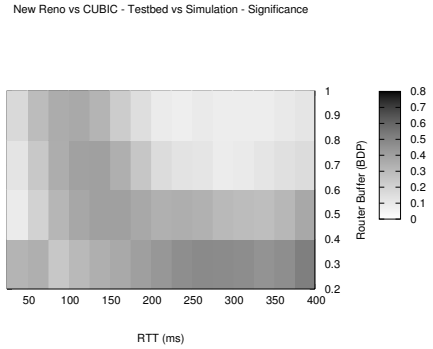

(c) Case *BI* - BIC



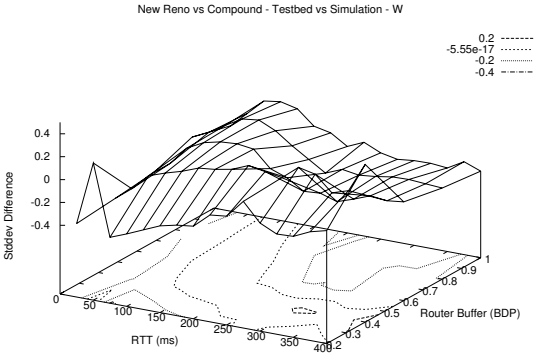

(d) Case *BI* - BIC



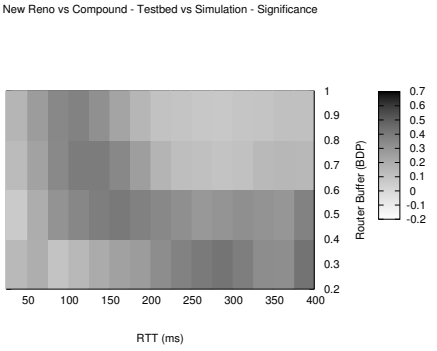

(e) Case *CU* - CUBIC




(f) Case *CU* - CUBIC




(g) Case *CO* - Compound TCP




(h) Case *CO* - Compound TCP

Fig. 5. Difference surfaces (W) for all cases

[3]. However, as our Primary contribution is a comparative analysis, this is not a concern. For considering the performance of Compound TCP, our Secondary contribution, the use of the JFI will show some bias, accentuating the unfairness of the TCP variants at high BDP, when their design goals are to perform better in such environments than TCP NewReno is capable of. We have previous work that describes a metric which overcomes this deficiency in the JFI [3], and we leave this analysis for further work.

## V. Summary, conclusion and future work

We have performed a simple set of experiments comparing several TCP variants, both in simulation and in a testbed in order to compare results between *ns2* and the testbed. We have deliberately kept the complexity of our network configuration low in order to highlight the TCP behaviour rather than explore 'realistic' network scenarios. We have also used an *ns2* extension to include the same TCP code into the simulation as is used by the testbed.

*We observe that if we consider the fairness using Jain's Fairness Index values in our pair-wise experiments, both simulation and testbed, there is a significant difference between results obtained from ns2 and from the testbed.*

The differences are not wholly consistent. Although there are similar regions of inconsistency across our experiments, we see that the results differ slightly in different regions of the parameter spaces (RTT values and RB values) depending on the protocol being tested. We find best consistency between *ns2* and the testbed when both RTT values are higher ($>\sim$200ms), and RB values are higher ($>\sim$0.6BDP). It is not clear from our experiments and analysis so far why these differences and inconsistencies between *ns2* and the testbed should be present: this paper has presented observation and analysis of the effects, and leaves to future work the investigation of reasons for their presence. However, given that the effects have now been observed, this means that *ns2* users can be aware of such artefacts when using TCP flows for the kind of comparative experiments and subsequent analysis as presented in this paper.

We find that BIC, CUBIC and Compound TCP all show unfairness to TCP NewReno when measured using Jain's Fairness Index. Fairness improves when both RTT increases and RB increases.

The output of the *iperf* runs, *ns2* runs, *netem* and *ns2* configuration scripts, and scripts used for processing the data are all available from the authors, on request.

### A. Future work

We have shown that *ns2* does not match our testbed case. Clearly, an issue for further investigation is *why* these differences exist. This would require looking at the operation of *ns2* and comparing it to the testbed.

As reported in other studies, router buffer sizing is still an open issue and although recent studies show increased understanding (e.g., [18]), there is perhaps still some further investigation on how it may affect end-system protocol behaviour as observed in our TCP experiments.


## Acknowledgements

The authors would like to express their appreciation for all those who have dedicated their time and energy to creating, maintaining and furthering the development of *ns2* , and those who have undertaken the task of creating *ns3*.


## References


[1] 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 *Proc. 11th Communications and Networking Simulation Symposium (CNS'08)*, Ottawa, Canada, April 2008.

[2] S. Bhatti, M. Bateman, and D. Miras. A Comparative Performance Evaluation of DCCP. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS2008), Edinburgh, UK*, June 2008.

[3] S. Bhatti, M. Bateman, D. Rehunathan, T. Henderson, G. Bigwood, and D. Miras. Revisiting inter-flow fairness. In *Proc. BROADNETS2008 - 5th International Conference on Broadband Communications, Networks and Systems*, London, UK, September 2008.

[4] Bob Briscoe. Flow rate fairness: dismantling a religion. *SIGCOMM Computer Communication Review*, 37(2):63–74, 2007.

[5] B. Even, Y. Li, and D. Leith. Evaluating the performance of TCP stacks for high-speed networks. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)*, Feb 2006.

[6] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), March 2008.

[7] S. Floyd and E. Kohler. Internet research needs better models. In *Proc.of HotNets–I*, October 2002.

[8] S. Floyd and E. Kohler. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control. RFC 4341 (Proposed Standard), March 2006.

[9] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 9:392–403, 2001.

[10] Y. Ganjali and N. McKeown. Update on buffer sizing in internet routers. *SIGCOMM Comput. Commun. Rev.*, 36(5):67–70, 2006.

[11] 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.

[12] J. Jaffe. Bottleneck Flow Control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[13] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. DEC Research Report TR-301, Sep 1984.

[14] 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*, volume 49, pages 237–252, 1998.

[15] S. Kunniyur and R. Srikant. End-to-end Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks. *IEEE/ACM Transactions on Networking*, 11(5):689–702, October 2003.

[16] D. Miras, M. Bateman, and S. Bhatti. Fairness of High-Speed TCP Stacks. In *22nd IEEE International Conference on Advanced Information Networking and Applications (AINA 2008)*, Ginowan, Japan, March 2008.

[17] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, 28(4):303–314, 1998.

[18] R. S. Prasad, C. Dovrolis, and M. Thottan. Router buffer sizing revisited: the role of the output/input capacity ratio. In *Proc ACM CoNEXT 2007*, pages 1–12, New York, NY, USA, 2007. ACM.

[19] R. N. Shorten and D. J. Leith. H-TCP: TCP for high-speed and long-distance networks. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)*, Feb 2006.

[20] 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.

[21] David X. Wei and Pei 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*, page 9, New York, NY, USA, 2006. ACM Press.

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

[23] 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.

## APPENDIX - PROTOCOL BEHAVIOUR

In this section, we briefly describe the behaviour of the individual protocols we will consider. Our description is intended to highlight the main features of each protocol.

Our selection of protocols is somewhat arbitrary: we have chosen protocols that are widely used (or are likely to be widely used), are readily available for use within our experimental platform, and are of interest to a user community as well as the research community.

### A. TCP NewReno

The basic congestion control algorithm in TCP NewReno is well known. 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 $\alpha$ segments when an ACKnowledgment is received, and decreases a factor $\beta$ 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.

### B. BIC

Binary Increase Congestion control TCP – *BIC* – 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 $\beta$. 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.

### C. CUBIC

*CUBIC* 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 $\beta$ is the multiplicative decrease factor after a loss event. $C$ and $\beta$ 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 $W_{max}$.

### D. Compound TCP

*Compound TCP* 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. The delay-based component 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, $\gamma$, 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:

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

where $k$, $\alpha$, $\beta$ and $\eta$ 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. $\gamma$ 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[11]. Compound TCP is also available for Linux. The implementation of Compound TCP used in our study is Caltech's Linux patch[12], 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.

[11] http://support.microsoft.com/kb/949316
[12] http://netlab.caltech.edu/lachlan/ctcp/