TCP versus TFRC over wired and wireless Internet scenarios: an experimental evaluation*

Luca De Cicco, Saverio Mascolo
ldecicco@poliba.it, mascolo@poliba.it

DEE Politecnico di Bari,Via Orabona 4, 70125 Bari, Italy

Abstract. TCP NewReno is the standard transport protocol originally designed to transport bulk data over the Internet. During the years it has been very successful to provide Internet stability due to its congestion control scheme. However TCP is not very suitable for multimedia streaming applications, that are time sensitive, because of its retransmission and multiplicative decrease mechanisms. The alternative to TCP is the User Datagram Protocol (UDP) which works as a simple packet multiplexer/demultiplexer and does not implement any congestion control scheme or retransmission mechanism. However, it has been pointed out that applications that don't use congestion control schemes are dangerous for the stability of the Internet [1]. The TCP Friendly Rate Control (TFRC) is currently been discussed within the IETF as a possible leading standard for streaming multimedia flows. This paper aims at investigating the performances of TCP and TFRC congestion control schemes in wired public Internet and in mixed wired/wireless Internet using a commercial UMTS card. The experiments carried out have shown that TFRC exhibits smoother rate dynamics in all wired scenarios, whereas in the case of UMTS scenario its burstiness is comparable to that of TCP.

1 Introduction

Nowadays Wireless Internet allows users to achieve ubiquitous access to the Internet. Moreover the new standards for broad band wireless networks such as IEEE 802.16, 802.16a, 802.11a/g and the new 3G UMTS networks enable users to access rich audiovisual contents.

TCP NewReno is the standard transport protocol originally designed to transport bulk data over the Internet, which has been very successful to provide Internet stability due to its congestion control scheme. TCP is not very suitable as a transport protocol for multimedia streaming applications because of its retransmission and multiplicative decrease features that are not useful with delay sensitive flows. The alternative to TCP is the User Datagram Protocol (UDP) which works as a simple packet multiplexer/demultiplexer and does not implement any congestion control scheme or retransmission mechanism. However it

* This work was supported by the MIUR-PRIN project no. 2005093971 "FAMOUS Fluid Analytical Models Of aUtonomic Systems"

has been pointed out that applications that don't use congestion control schemes to adapt their rate in order to avoid congestion collapse, are dangerous for the stability of the Internet [1]. Many efforts have been carried out to the purpose of designing new end-to-end protocols able to efficiently stream multimedia flows over wired/wireless scenarios and to assure network stability such as TFRC [2] and RAP [7]. When a new protocol is proposed it has to satisfy the following requirements: i) the rate of generated flows should be smooth, i.e. rates should exhibit contained oscillations in order to keep the receiver buffer as small as possible; ii) it has to be TCP friendly i.e. competing TCP flows should gain similar long term throughput; iii) it has to be fair i.e. flows using the same congestion control should gain the same long term throughput; iv) it has to be responsive i.e. flows should quickly react to network condition changes.

The TCP Friendly Rate Control (TFRC) is currently discussed within the IETF as a possible leading standard for streaming multimedia flows [2].

This paper aims at investigating the performances of TCP and TFRC congestion control schemes in wired public Internet and in mixed wireless/wired Internet using a commercial UMTS card.

The paper is organized as follows. In section 2 we will briefly describe TCP Reno congestion control and TFRC basics. Section 3 describes the tools we developed and used to collect experimental analysis; moreover we describe the testbed used in our experiments. In section 4 we report results obtained over both wired Internet and using a commercial UMTS card provided by a telecom operator. Section 5 reports burstiness indices measures for each scenario we tested. In the final section we report conclusions and open issues.

2 TCP and TFRC congestion control basics

The version of the TCP (TCP NewReno) congestion control algorithm which is currently implemented in TCP/IP stacks is largely based on [6] and on its modifications. TCP congestion control is made of two main different phases: the probing phase and the decreasing phase. In the probing phase the channel is probed by exponentially increasing the congestion window (*slow start phase*) until the slow start threshold `ssthresh` is hit. At this point the congestion window `cwnd` is linearly increased (*Additive Increase* or *congestion avoidance* phase).

The decreasing phase, also called *Multiplicative Decrease*, is instead triggered when a congestion episode is experienced. TCP assumes that a congestion takes place when three duplicate acknowledgment packets (3DUPAK) are received by the sender or a timeout expires. When such an event occurs the congestion window is halved in order to quickly react to the congestion episode.

The pseudo code of TCP according to [3] is the following:

1. On ACK reception:
   - *cwnd* is increased according to the Reno algorithm
2. When 3 DUPACKs are received:

- `ssthresh = max(FlightSize/2, 2);`
- `cwnd = ssthresh;`

3. When coarse timeout expires:
   - `ssthresh = 1;`
   - `cwnd = 1;`

One of the main drawbacks of classic TCP congestion control is experienced when accessing lossy links such as 802.11b/g and 2G/3G network. In fact TCP triggers the Multiplicative Decrease even if the loss is due to interference on the wireless channel and not to congestion.

The TCP Friendly Rate Control (TFRC) is a rate based congestion control which aims at obtaining a smooth rate dynamics along with ensuring friendliness towards Reno TCP [5]. To provide friendliness, a TFRC sender emulates the long term behavior of a Reno connection using the equation model of the Reno throughput developed in [4]. In this way, the TFRC sender computes the transmission rate as a function of the average loss rate, which is sent by the receiver to the sender as feedback report.

3 Experimental testbed and measurement tools

Figure 1 shows the wired and wireless scenarios we tested. In both the wired and wireless scenarios TCP and TFRC senders were located at University of Uppsala (Sweden), whereas the receivers were located in Bari (Italy) accessing the wired Internet through a host located at Politecnico di Bari or the public wireless Internet on UMTS from a commercial telecom operator. The reverse traffic flows have been generated running TCP or TFRC senders at Politecnico di Bari and TCP or TFRC receivers at Uppsala.

In order to generate TCP flows we used `iperf` which has been modified to incorporate `libnetmeas` (see below) and to automatically produce log files. As for the TFRC flows we used experimental code at sender and receiver side [12]. We have no control of other competing flows on the network as we evaluated protocols over the public Internet so we can't isolate them from other flows. In each scenario we have measured goodputs, fairness indices and burstiness indices we report in the following sections.

When conducting and collecting live Internet experiments one of the most difficult task is to log TCP variables such as *congestion window, slow start threshold, round trip time* and so on. Since TCP is implemented in the kernel of the operating system those variables are kept hidden to user space application making their logging from the user space an impossible task. In order to work around this issue researchers have proposed several solutions: instrumenting the kernel code, developing TCP user space implementation [9] and using packet sniffers along with `tcptrace` application. Each of these solutions are not well suited because it is difficult to validate instrumented implementations and it is even more difficult to verify a user space TCP implementation. Using a packet sniffer is not

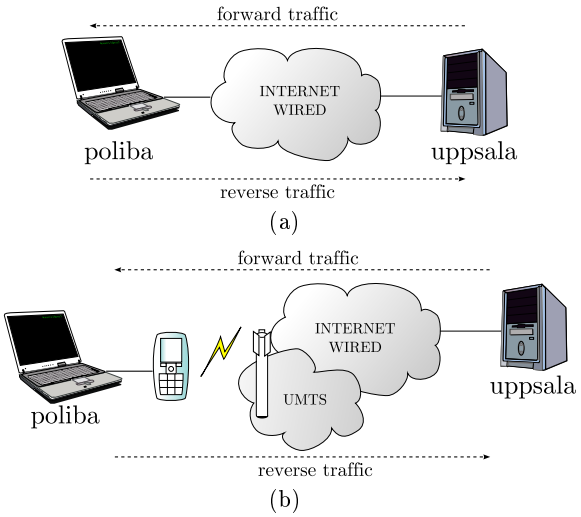


Fig. 1. Experimental testbed for wired (a) and wired/wireless (b) scenarios

suitable as well because sniffed packets don't contain any information about the TCP internal state.

In [8] authors describe a new and less intrusive solution which consists of a kernel patch and a library (`libweb100`) which exposes to the user space variables of each TCP flow. The interface between the kernel-space and the user-space is the virtual *proc* filesystem where statistics about flows are kept. Each flow is associated to a file in `/proc/web100/CID` where CID is a number incremented on the establishment of a new TCP flow. In order to log a TCP flow, it is necessary to know the CID and then it is possible to use one of the web100 tools (i.e. `readvars`) in a loop. This is a really difficult task because the CID is not known and the user has to manually find the CID matching the right connection.

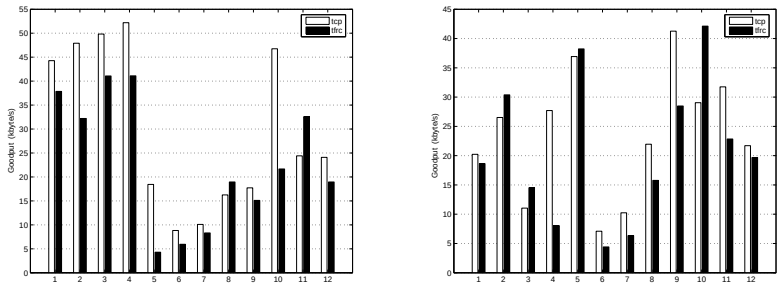
In order to log TCP flows we developed a library which is able to overcome the aforementioned issues [11]. The library depends on `libweb100`, it is written in C language using `glib` and it is shipped with a very simple API (Application Program Interface) in order to be easily integrated in existing applications. The library is initialized using a function of the API which starts an internal thread that will automatically log flows matching a string creating or using a specified socket in a file called *tcp_<CID>_<timestamp>.txt* where CID is the connection ID and `timestamp` is the UNIX timestamp of the first data logged. Moreover the API offers a way to select the congestion control algorithm to use. The integration of the library in an existing application is really trivial: the application must call the initialization function on the ports (or on the socket) it wants to listen on and must link the library to the application (see library documentation for further details).

4 Experimental results

4.1 Experiments over wired Internet

In this section we report results obtained testing TCP and TFRC in the following scenarios (see Figure 1 (a)): i) single TCP vs single TFRC flow without reverse traffic; ii) Single TCP vs single TFRC flow with reverse traffic; iii) Single TCP vs single TFRC flow with parallel UDP connection; iv) 3 TCP flows vs 3 TFRC flows without reverse traffic.
In all tests the receiver is located at Politecnico di Bari, Italy and the sender is located at University of Uppsala, Sweden. In each of the considered scenario we will report goodputs, and instantaneous throughput of the most interesting experiments. In each test we run consecutively TCP and TFRC flows for two minutes in order not to have inconsistent results due to different network conditions.

Single TCP vs Single TFRC flow without reverse traffic. Figure 2 (a) shows goodput achieved by TCP and TFRC flows that were run at different times and in different days. It is worth noticing that in 10 out of the 12 tests TCP achieves higher throughput with respect to TFRC. In some experiments (number 5 and number 10) the gap between TCP and TFRC is noticeable.

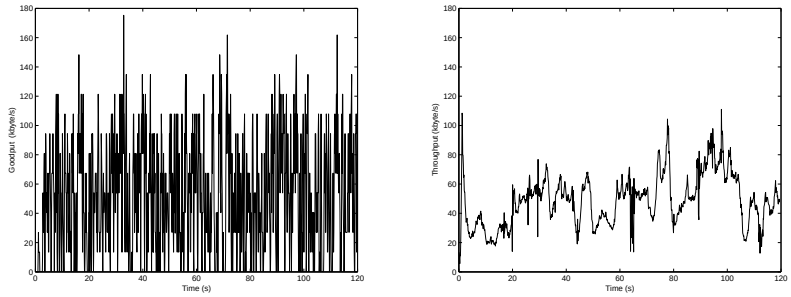


(a) (b)

Fig. 2. Goodputs of single TCP vs single TFRC flow (a) without reverse traffic (b) in the presence of reverse traffic

Figures 3 (a) and (b) show instantaneous throughput of a TCP and a TFRC connection. By comparing the two figures we can conclude that TFRC flows are smoother than TCP flows in this scenario but TFRC achieves a lower goodput than TCP.

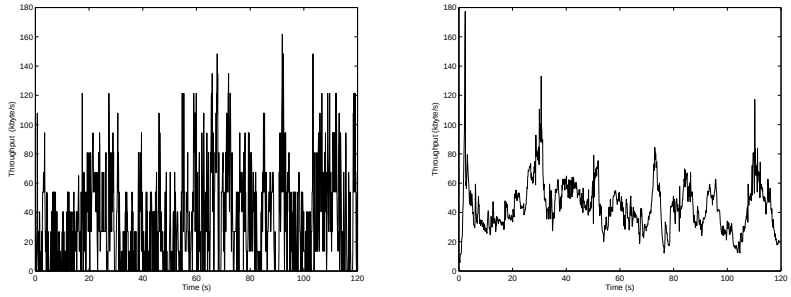
Single TCP vs single TFRC flow with reverse traffic. In this scenario we evaluate TCP and TFRC performances in the presence of a TCP flow in the



(a) (b)

Fig. 3. Instantaneous throughputs of (a) TCP; (b) TFRC

backward path. Figure 2 (b) shows the TCP and TFRC achieved throughputs. By comparing Figure 2 (a) and (b) we can see that the TCP goodput is sensitive to the congestion on the backward path whereas TFRC is not due to the fact that feedback reports are much less frequent than ACK packets (one packet every RTT). In the considered scenario TCP flows achieve a higher throughput with respect to TFRC in 6 tests over 12.



(a) (b)

Fig. 4. Instantaneous throughputs of: (a) TCP; (b) TFRC (b)

Figures 4 (a) and (b) show instantaneous throughputs for a TCP and a TFRC connection. Also in this scenario TFRC exhibit a smoother throughput dynamics respect to TCP.

Single TCP vs single TFRC flow with parallel UDP connection. In this scenario we report results obtained testing TCP and TFRC with one concurrent UDP flow generated by the Uppsala Host in order to force a 20 Kbyte/s

bandwidth limitation at the receiver. Figure 5 shows goodputs for the TCP and TFRC flows: the TCP provides better link utilization.

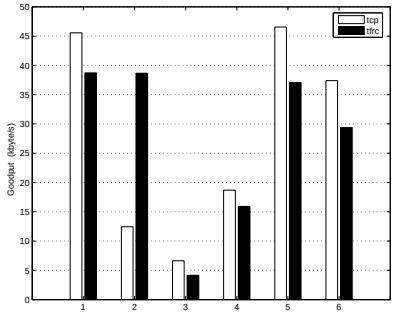
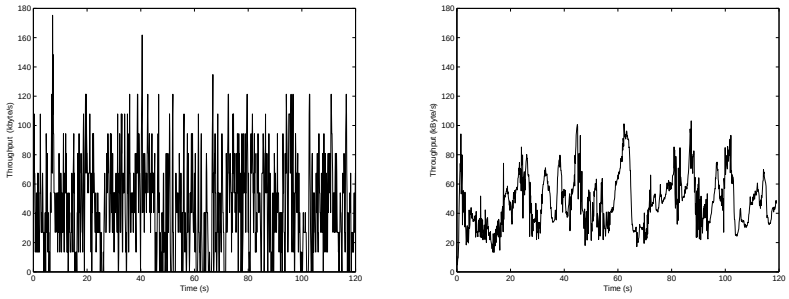


Fig. 5. Single TCP vs Single TFRC flow with parallel UDP connection

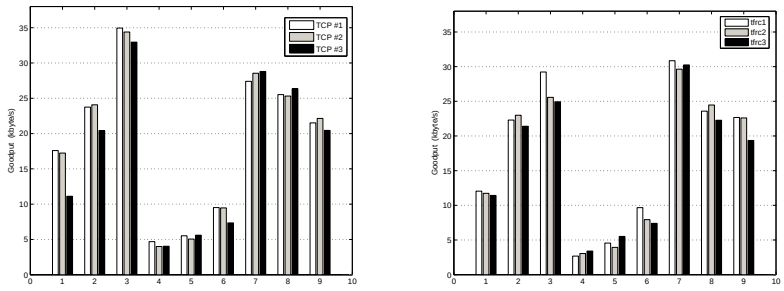
Figures 6 (a) and (b) depict the dynamics of TCP and TFRC instantaneous throughputs respectively in the considered scenario. Again, TFRC exhibit a smoother dynamics with respect to TCP.



(a) (b)

Fig. 6. Instantaneous TCP (a) and TFRC (b) goodput

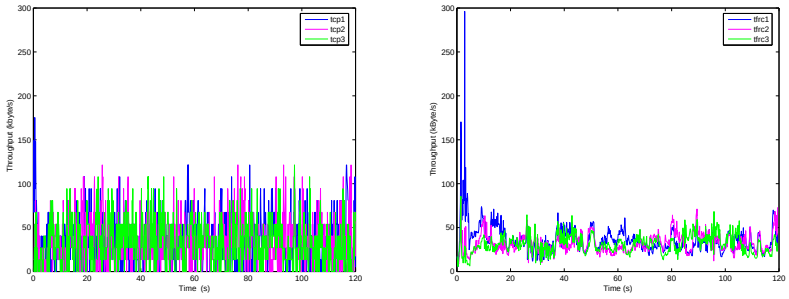
Three TCP flows vs three TFRC flows without reverse traffic. In this scenario we test intraprotocol friendliness using the Jain Fairness index [13] when three TCP flows or three TFRC flows share the same link. Figures 7 (a) and (b) report the TCP and TFRC goodputs respectively.



(a) (b)

Fig. 7. Goodput for: (a) 3 TCP flows; (b) 3 TFRC flows without reverse traffic

Figures 8 (a) and (b) depict the behaviour of TCP and TFRC instantaneous throughput of all flows in the considered scenario. Again, TFRC exhibit a smoother dynamics with respect to TCP.



(a) (b)

Fig. 8. Instantaneous TCP (a) and TFRC (b) goodput

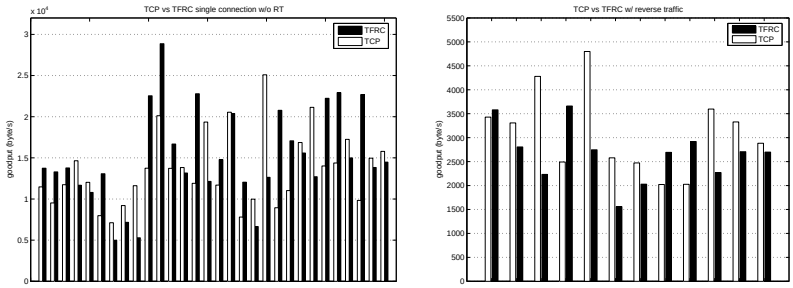
The Jain Fairness index has been evaluated. Each protocol achieves a JF index near to 1, which is the maximum possible value for the index.

4.2 Experiments over the UMTS link

Single connection without reverse traffic. In this section we describe results obtained when a single TCP or TFRC connection uses an UMTS downlink. Figure 9 (a) depicts throughput as measured at the receiver. It is worth noticing that, nor TCP neither TFRC, reach the nominal downlink capacity of 384 $Kbps$.

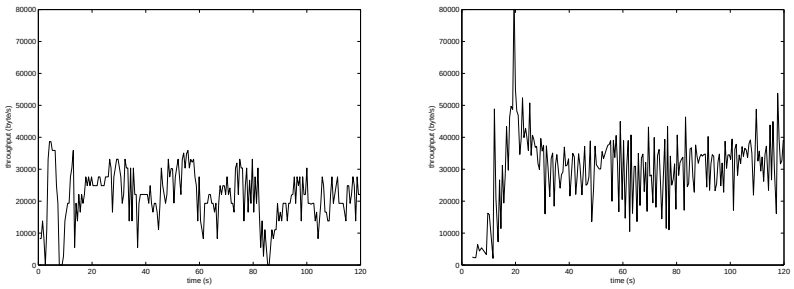
Figure 10 depicts the TCP and TFRC received throughput during two consecutive tests. Both protocols show remarkable oscillations in throughput. Moreover

it is worth noticing that the TFRC transient is long (approximately $20s$) if compared to the TCP transient time. It seems that using TFRC for video streaming in UMTS scenarios would require a longer buffering phase if compared to TCP behaviour. Moreover, by comparing Figure 3 (a) and Figure 10 (a) we can observe different behaviour of TCP in the wired and wireless scenario respectively. In fact in the UMTS scenario the TCP burstiness is clearly mitigated and it is comparable to that of TFRC. We have obtained similar results by repeating the experiments many times over different days.



(a) (b)

Fig. 9. TCP vs TFRC goodputs (a) without reverse traffic (b) with reverse traffic



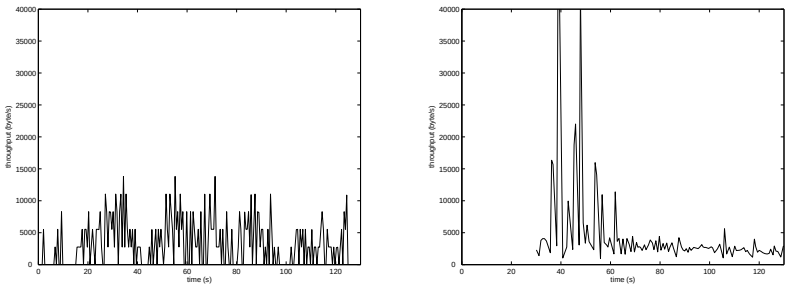
(a) (b)

Fig. 10. Instantaneous throughput of (a) TCP and (b) TFRC without reverse traffic

Single connection with reverse traffic. In this scenario the TCP and TFRC have been tested in presence of homogeneous reverse traffic in order to evaluate if the protocols are sensitive to congestion on the backward path. For what

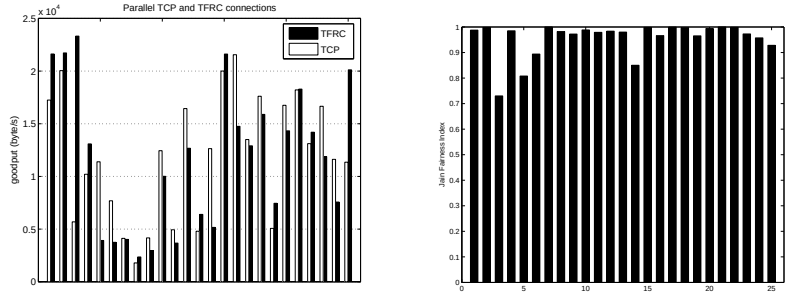
concerns TCP, we run iperf in bidirectional mode on both the UMTS clients in Bari and the wired host at Uppsala, whereas to test TFRC in this scenario we run both the sender and receiver on UMTS client in Bari and on Uppsala client.

By comparing Figure 9 (b), which shows goodputs in the present scenario, and Figure 9 (a), we can notice that goodputs suffer a dramatic drop when using TCP or TFRC in presence of reverse traffic. This results are quite disappointing if an UMTS connection has to be used in a peer to peer system when a bidirectional communication is set up. Figure 11 reports instantaneous rates of a TCP and a TFRC flow. Both TFRC and TCP provide a very low link utilization in the presence of reverse traffic (around 25 $Kbps$ on average).



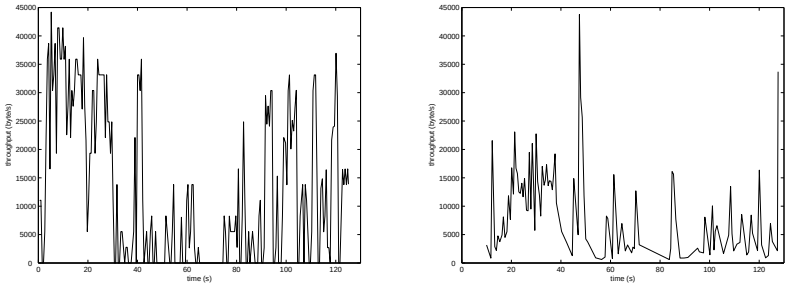
(a) (b)

Fig. 11. Instantaneous throughput of (a) TCP and (b) TFRC in the presence of reverse traffic



(a) (b)

Fig. 12. TCP and TFRC accessing the same link; (a) goodputs, (b) Jain Fairness index.



(a) (b)

Fig. 13. TCP (a) and TFRC (b) instantaneous throughput when simultaneously accessing the link

One TFRC flow and one TCP flow sharing the downlink. Here we collect results obtained when one TCP and one TFRC flow share the UMTS downlink. Examining Figure 12 (a) we can notice that the throughput of each connection is not affected from the other flow and the downlink capacity is not underutilized. In order to produce a quantitative measurement of the inter-protocol fairness we evaluated the Jain Fairness Index. Figure 12 (b) shows fairness indices which are near to the maximum value of 1 in most of the tests. Figure 13 shows instantaneous throughputs of a TCP flow and a TFRC flow which simultaneously access the UMTS link. It is worth noticing that even if the channel utilization is quite good each flow exhibits pronounced oscillations.

5 Burstiness of received data

In order to evaluate the smoothness of the data transfer rate we have evaluated the burstiness index of each transfer. The burstiness index is defined as $b = \frac{\sigma(r)}{E[r]}$ where $\sigma(r)$ represents the standard deviation of the received rate r and $E[r]$ is the average value [14].

Figure 14 shows that TFRC halves the burstiness index with respect to TCP in all wired scenarios. However in the UMTS scenario TCP and TFRC provide similar burstiness indices, except in the case with reverse traffic where TCP is less bursty than TFRC.

6 Conclusions

We conducted several experiments testing TCP and TFRC behaviour both in wired and UMTS networks measuring goodputs and fairness indices. TFRC exhibits smoother rate dynamics in all wired scenarios, whereas in the case of UMTS scenario its burstiness is comparable to that of TCP. Moreover experimental results have shown that when TFRC and TCP flows are accessing an

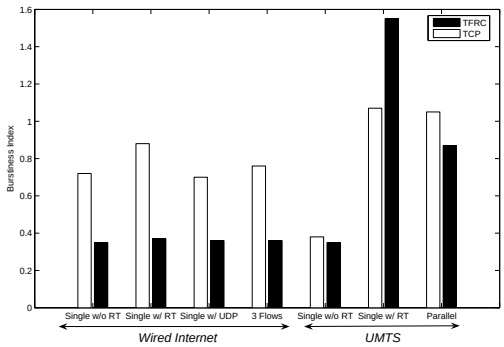


Fig. 14. Burstiness Indices

UMTS link they both are not able to provide full link utilization in the presence of reverse traffic, which could be a severe limitation in peer-to-peer applications.

References

1. S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet", *IEEE/ACM Transaction on Networking*, vol. 7, no. 4, pp. 458-472, 1999.
2. M. Handley,S. Floyd, J. Padhye and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", *RFC 3448*, January 2003.
3. M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", *RFC 2581*, April 1999.
4. J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. "Modeling TCP throughput: A simple model and its empirical validation", ACM Sigcomm '98, pages 303-314, Vancouver BC, Canada, 1998.
5. S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-based congestion control for unicast application", ACM SIGCOMM 2000, August 2000
6. V. Jacobson, "Congestion Avoidance and Control", ACM Computer Communications Review, 18(4): 314 - 329, August 1988.
7. R. Rejaie, M. Handley, D. Estrin, "RAP: An End-to-end Rate-based congestion control mechanism for Real-time streams in the Internet", Proc. of INFOCOM, March 1999
8. M. Mathis, J Heffner and R Reddy, "Web100: Extended TCP Instrumentation for Research, Education and Diagnosis", ACM Computer Communications Review, Vol 33, Num 3, July 2003.
9. T. Dunigan, F. Fowler, "A TCP-Over-UDP test Harness", Technical report
10. Iperf, http://dast.nlanr.net/Projects/Iperf/
11. LIBrary for NETwork MEASurements, http://c3lab.poliba.it/index.php/Libnetmeas
12. TFRC experimental Code, http://www.icir.org/tfrc/code/
13. R. Jain, "The art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation and Modeling", John Wiley and Sons, April 1991.
14. YR. Yang, MS. Kim, S. Lam , "Transient behaviors of TCP- friendly congestion control protocols.", Proc. IEEE INFOCOM 2001, April 2001