Streaming over Edge/UMTS commercial cards using TCP and TFRC: an experimental evaluation*

Luca De Cicco, Saverio Mascolo

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

Abstract Streaming videos over mobile phones is an emerging application. The TCP Friendly rate control is currently discussed within the IETF as a possible leading standard for streaming time-sensitive data such as audio and video over the Internet. The main feature of TFRC is the rate control that is aimed at smoothing the burstiness of TCP due to its window control. In this paper we evaluate TCP versus TFRC using a commercial Edge/UMTS card connected to the public Internet. We evaluate and compare inter-protocol friendliness, smoothness and googdput of TCP versus TFRC. To the purpose a library has been developed on top of the Web100 tools which enables, through the use of a simple API, to log TCP variables in any application which generates TCP flows. Main finding of this investigation is that both TCP and TFRC are not able to use downlink bandwidth in the presence of reverse traffic, which is an important working condition such as in the case of peer-to-peer VoIP applications.

1 Introduction

Audiovisual streaming is an emerging application in the odiern Internet. Applications that generate multimedia flows often do not use congestion control schemes to adapt their rate in order to avoid congestion collapse and assure internet stability. Several congestion control protocols have been proposed in order to transport multimedia flows. A new trasport protocol 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 the same 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 conditions changes.

TFRC [1] is currently been discussed within IETF as a possible leading standard for streaming multimedia flows.

In this paper evaluate TFRC performances versus TCP by accessing the public Internet using a commercial UMTS card.

* The work has been supported by Financial Tradeware Srl, Roma, Italy

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 for experimental analysis . In section 4 we describe the testbed used in our experiments and we report results obtained by live internet measurement using a commercial UMTS card provided by a telecom operator. Final section reports conclusions and open issues.

2 Related Work

The version of the TCP (TCP NewReno) congestion control algorithm which currently implemented in TCP/IP stacks is largely based on [2] 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) untill the slow start threshold is hit. At this point the congestion window 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 is the following:

1. On ACK reception:
   - *cwnd* is increased according to the Reno alogrithm
2. When 3 DUPACKs are received:
   - `ssthresh = max(2,cwnd/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 netwok. 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 algorithm which has been recently proposed within IETF as the standard transport protocol for multimedia flows. TFRC aims at obtaining a smooth rate dynamics along with ensuring friendliness towards Reno TCP [4]. 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 [3]. 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 Issues in measuring TCP flows performances

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 an impossible task. In order to work around this issue researchers have proposed several solutions: instrumenting the kernel code, developing TCP user space implementation [6] and using packet sniffers along with `tcptrace` application [7]. 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.

In [5] 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 kernel-space and user-space is the virtual *proc* filesystem where statistics about flow are kept. Each flow is associated to a file in `/proc/web100/CID` where CID is a number incremented on the estabilishment 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.

WAD (Work Around Daemon) [8] is a daemon which depends on libweb100. Is is written in python language and logs variables matching a defined pattern. However WAD is a standalone application and it can't be integrated in existing applications (say `iperf`) in order to have a self contained tool which automatically produces log files.

Here we propose a library we developed which is able to overcome the aforementioned issues. 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.

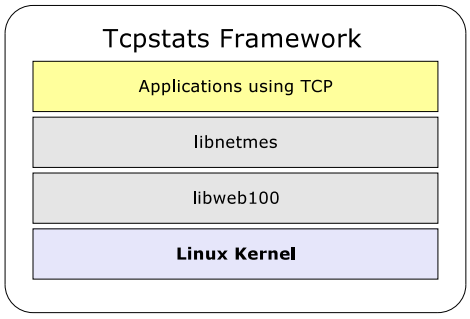


Figure 1. TCP Stats framework

In order to use the library it is sufficient to use the following C function:

```
gboolean init_tcpstats(const gchar *lport_token, gint sfreq);
```

This function initializes the tcpstats internal thread which automatically logs all flows which match the `lport_token` string creating a file *tcp_<CID>_<timestamp>.txt* where CID is the connection ID and `timestamp` is the UNIX timestamp of the first data logged. `lport_token` is the local port to listen, a range of ports or a list of ports (for further details see the documentation). Moreover it is possible to set the congestion control algorithm using the following function:

```
set_congestion_control (gint cong)
```

Integrating the library in an existing applitation is as easy as calling `init_tcpstats()` and linking the application to `libnetmes` (see library documentation for further details).

We used `libnetmes` in our tests after integrating it in `iperf`. In this was we provide a complete tool which automatically logs its generated TCP flows. We believe that promoting the use of this library will allow researchers to produce homogeneous logs. The architecture measurement Framework architechture is depicted in Figure 1.

4 Testbed, scenarios and results

In order to generate TCP flows we used `iperf` which has been modified to incorporate `libnetmes` and to produce logs automatically. As for the TFRC flows we used experimental code [11] at sender and receiver side.

The sender is located at university of Uppsala (Sweden) and the receiver, which is located in Bari, Italy, is accessing the public Internet using a commercial UMTS card by TIM. Each protocols have been tested (see figure 2) in three different scenarios: i) single connection without reverse traffic; ii) single connection with reverse traffic; iii) one TFRC flow sharing the link with one TCP flow. We have measured goodputs and burstiness.

The burstiness index [13] has been measured using the following index of burstiness:

$$b = \frac{\sigma(r)}{E[r]} \tag{1}$$

where $\sigma(r)$ represents the standard deviation of the received rate and $E[r]$ is the average value.

In the following subsections we report all results obtained carrying out the tests.

4.1 Single connection without reverse traffic

In this section we describe results obtained when a single TCP or TFRC connection uses the UMTS downlink. Figure 3 depicts received goodput measurement

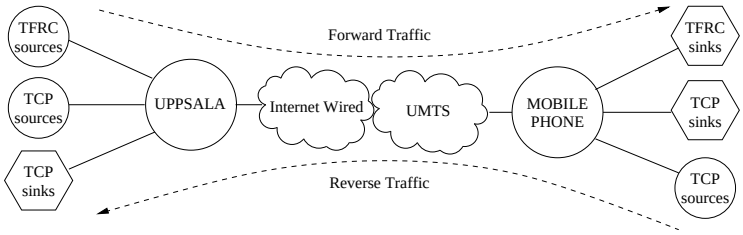


Figure 2. Experimental testbed

of each test. It is worth noticing that, nor TCP neither TFRC, reach the nominal downlink capacity of 384 $Kbps$. This is an expected behaviour for TCP flows over wireless link as long as its congestion control can't distinguish between losses due to congestion episodes or due to wireless channel interference. Similar results are obtained using TFRC congestion control algorithm.

Figure 4 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.

We have obtained similar results by repeating the experiments many times over different days.

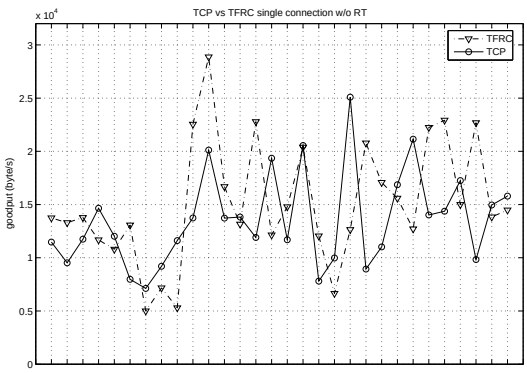
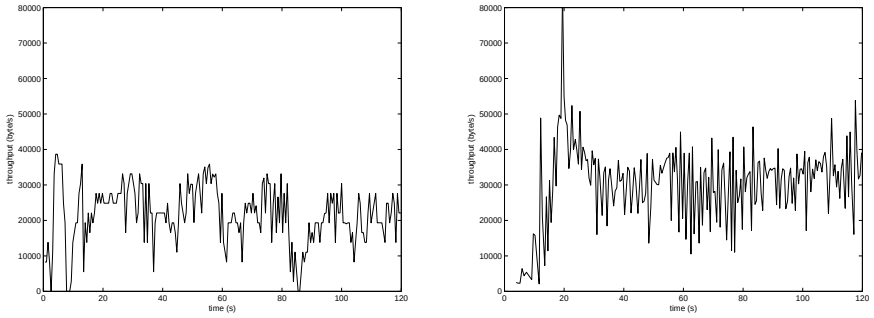


Figure 3. TCP vs TFRC goodputs without reverse traffic



(a) (b)

Figure 4. TCP (a) or TFRC (b) istantaneous throughput without reverse traffic

4.2 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 client in Bari and at Uppsala, whereas to test TFRC in this scenario we run sender and receiver on UMTS client and Uppsala, respectively.

By comparing Figure 5, which has been obtained measuring goodputs in the present scenario, and Figure 3, 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 such as in the case of VoIP applications where a bidirectional communication is set up.

In Figure 6 istantaneous rates of a TCP and TFRC flows are reported. Results obtained here are consistent to what have been shown about TCP behaviour in presence of the *ack compression* phenomena. Moreover TFRC exhibits a very long transient period and a very low link utilization in the presence of TFRC reverse traffic.

4.3 One TFRC flow and one TCP flow sharing the downlink

Here we collect results obtained when one TCP and one TFRC flow are present on the UMTS downlink. Examinating Figure 7 (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 [12],

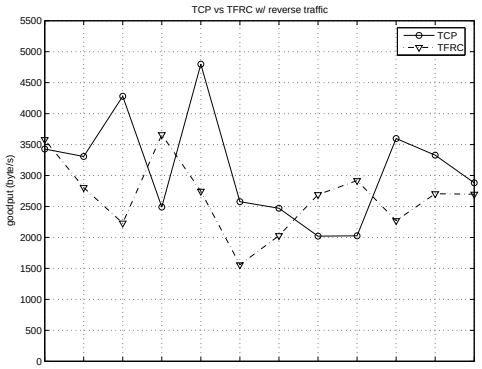
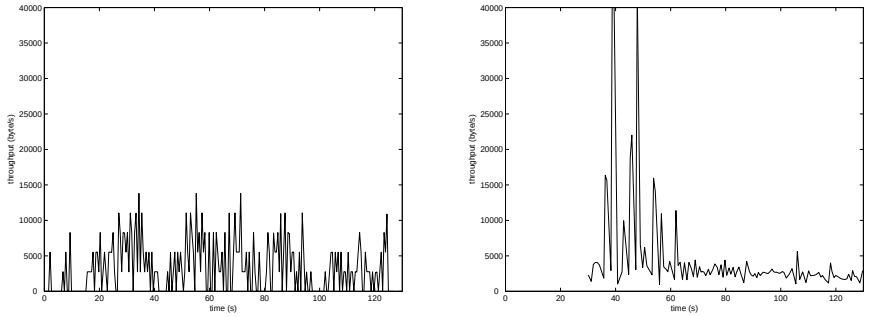
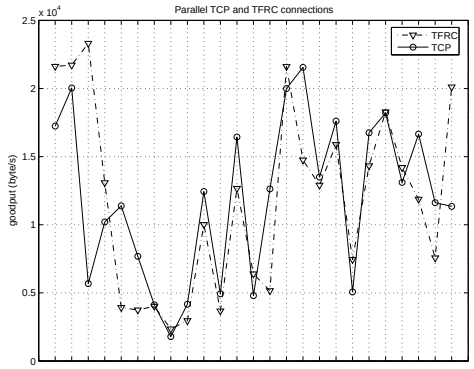


Figure 5. TCP vs TFRC goodputs with reverse traffic

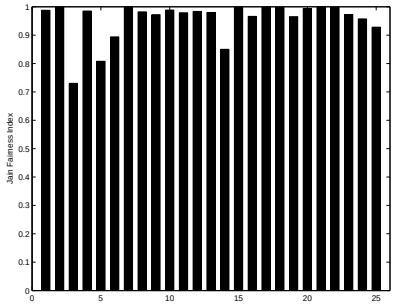


(a) (b)

Figure 6. TCP (a) vs TFRC (b) istantaneous throughput in the presence of reverse traffic

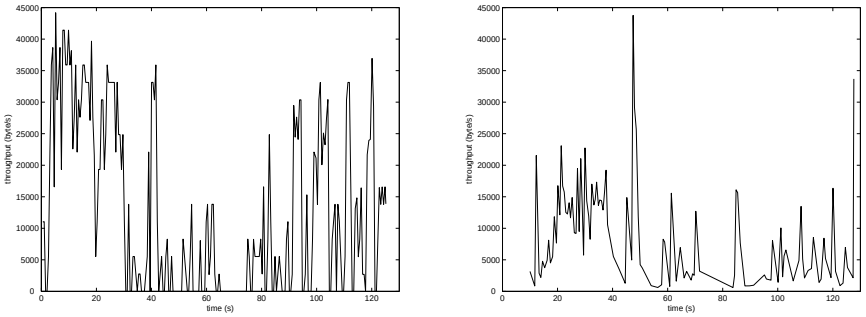


(a)



(b)

Figure 7. TCP and TFRC accessing the same link (a). Jain fairness index (b)



(a) (b)

Figure 8. TCP (a) and TFRC (b) istantaneous throughput when simultaneously accessing the link

defined as follows:

$$J_{FI} = \frac{\left(\sum_{i=1}^{n} b_i\right)^2}{n \sum_{i=1}^{n} b_i^2}$$

where n is the number of flows on the link and b_i is the goodput achieved by i^{th} connection. Figure 7 (b) shows fairness indices which are near to the maximuma value of 1 in most of the tests.

Figure 8 shows istantaneous throughput 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.

4.4 Burstiness evaluation and mean goodputs

Here we summarize values of the burstiness indices evaluated using (1) for selected flows in each scenarios. As we can see looking at Figure 9, burstiness index values are similar for each protocol in each of the three scenarios we have tested. In other terms TFRC doesn't produce smoother flows compared to TCP flows in each of the tested scenarios. Moreover both protocols behave better when accessing the UMTS downlink without reverse traffic. The worst condition is experienced when the backward path is congested.

Figure 10 shows average goodputs achieved by TFRC and TCP in considered scenarios.

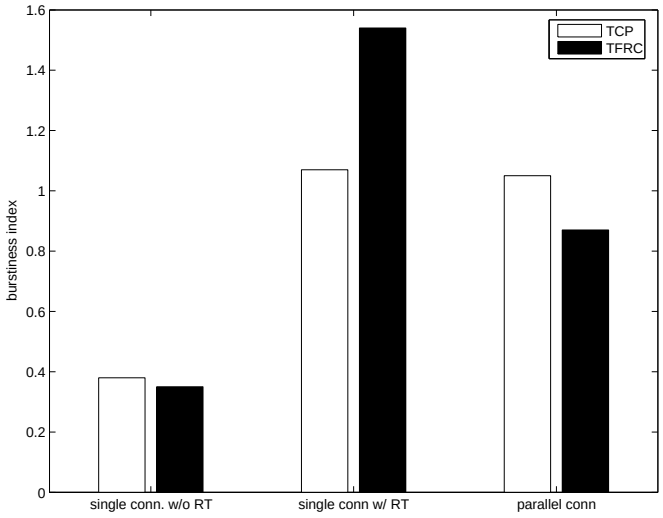


Figure 9. Burstiness indices

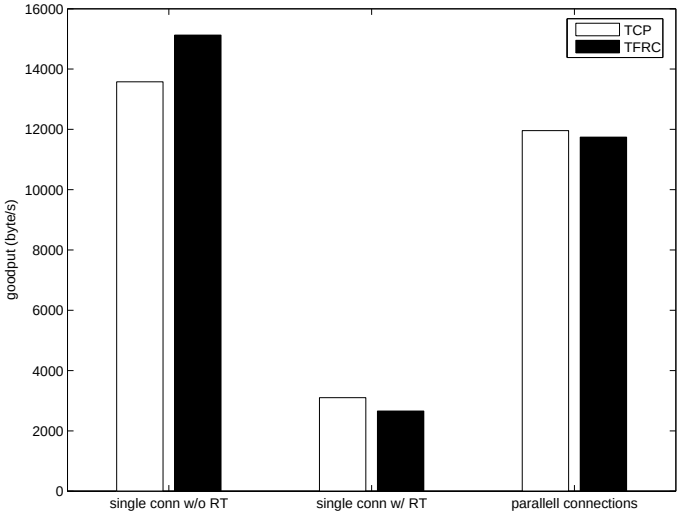


Figure 10. Average goodput in three different scenarios

5 Conclusions

We have tested TCP and TFRC congestion control algorithm when accessing the public Internet using a commercial UMTS card. The purpose was to evaluate their performances in 3G systems when streaming multimedia flows.

Results have shown that goodput provided by TFRC or TCP are similar, whereas burstiness is different in only one test scenario. Main problems we have found is that both TFRC and TCP are not able to provide UMTS link utilization when in the presence of reverse traffic that can be a severe limitation in P2P VoIP applications.

Finally, TFRC burstiness has been proved to be comparable to TCP burstiness in each scenario so both protocols requires quite the same receiver buffer size to cope with oscillation in received rate.

6 Acknowledgments

We tank Prof. Marco Ajmone Marsan who generously supported the start of this investigation.

References

1. M. Handley,S. Floyd, J. Padhye and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 3448, January 2003.
2. V. Jacobson, "Congestion Avoidance and Control", ACM Computer Communications Review, 18(4): 314 - 329, August 1988.
3. 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.
4. S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-based congestion control for unicast application", ACM SIGCOMM 2000, pages 43-56, Stockholm, Sweden, August 2000
5. 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.
6. T. Dunigan, F. Fowler, "A TCP-Over-UDP test Harness", Technical report
7. S. Ostermann. TCPtrace, 2003. http://www.tcptrace.org/
8. Dunigan, T., M. Mathis and B. Tierney, "A TCP Tuning Daemon , Proceeding of IEEE Supercomputing 2002 Conference", Nov. 2002, LBNL-51022.
9. iperf: http://dast.nlanr.net/Projects/Iperf/
10. L. De Cicco, libnetmeas, http://193.204.59.123/c3lab/libnetmeas.php
11. J. Widmer, "TFRC experimental Code", http://www.icir.org/tfrc/code/
12. R. Jain, "The art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation and Modeling", John Wiley and Sons, April 1991.
13. Yang YR, Kim MS, Lam S (2001) "Transient behaviors of TCP- friendly congestion control protocols." In: Proceedings of IEEE INFOCOM 2001, Anchorage, AK, April 2001, pp 22-26