Variable Packet Size Equation-Based Congestion Control[*]

Pedro Reviriego Vasallo[+]
International Computer Science Institute (ICSI)
1947 Center Street, Berkeley, 94704 California (USA)

Abstract

This paper, extends previous work [1-3] in equation-based congestion control for unicast traffic. Most best effort traffic on the internet is appropriately served by TCP which is the dominant transport protocol on the internet. However, there is a growing number of multimedia applications for which TCP is not well suited. For those applications, several congestion control mechanisms have been proposed [1] in order to avoid congestion collapse on the internet [4]. One of them is the recently proposed TCP Friendly Rate Control Protocol (TFRC) [1-3]. It can be only used by flows that have a constant packet size. In this paper, we propose an extension to the TFRC protocol in order to support variable packet size flows. Variable packet size has been used for the transmission of video over the internet [5] and is also used in voice applications. So it is important for a congestion control protocol to support variable packet size flows.

We also explore the concept of fairness among flows when some of the flows send small packets. Currently, these flows are penalized by TFRC because it imitates TCP's behavior giving less throughput to flows that use small packets. We argue that if a flow is sending small packets because the application requires it to do so (for example to minimize delay in a voice over IP conversation) it should get the same amount of bandwidth as a TCP session using large packets. This results in a modified concept of TCP friendliness that we introduce in this paper.

Finally we analyze some shortcomings of the equation used by TFRC to model TCP behavior and show that the impact of TCP timeouts are not completely modeled by the current TFRC equation.

Introduction

The congestion control mechanism used by TCP reacts to a single packet loss by halving its congestion window. This causes abrupt changes in the sending rate that are not appropriate for multimedia flows that require a smooth variation of the sending rate.

In Equation-based congestion control algorithms, the sender uses a equation that gives the maximum acceptable sending rate as a function of the loss rate. In this way the variations of the sending rate over time are smoother giving a better support for Continuous Media (CM) applications. If we want to compete fairly with TCP, we should use a equation that approximates the behavior of TCP.

[*] This work was supported by the Comisión Interministerial de Ciencia y Tecnología of Spain under the ICSI visitor programme.
[+] Currently with Teldat S.A. (previriego@teldat.es).

TFRC uses the following equation to model TCP behavior:

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)} \qquad (1)$$

Where T is the sending rate, s is the packet size, R the round trip time, t_{RTO} the TCP retransmit timeout value and p the loss event rate as defined in [1].
It is clear from equation (1) that TFRC can not be used for variable packet size flows.

In some CM applications there is a need for sending data with different packet sizes. For example, suppose that we are coding a speech signal using the G. 729 Annex B speech coder [6] and that we are sending the voice frames over IP using the Real Time Protocol (RTP) [7] and encapsulating ten voice frames in an IP packet. Whenever we find a silence frame we have to stop inserting voice frames into a packet and send it right away. At that time we may have only one or two frames so the IP packet will have a different size. The use of different packet sizes has also been proposed for error-resilient video transmission over the internet [5].

We also note that for the example of the voice flow that we have just discussed, the packets will be around 140 bites long. When we apply equation (1) to this flow and to a flow that is sending 1400 packets (assuming that the rest o the parameters are the same) we see that the voice flow gets ten times less bandwidth than the other flow. This is because TCP flows using small packet sizes get less bandwidth than TCP flows using large packet sizes. Normally, TCP flows use large packet sizes if they have data available to be transmitted, which is usually the case in FTP or Web transfers. But some CM applications can not send large packet sizes because that would produce an excessive delay.

In the rest of the paper we extend TFRC to support variable packet size flows and we also propose to modify equation (1) to make it independent of the packet size so that flows that are forced to send small packets by the nature of the application are not penalized.

Fairness Revisited

Equation (1) approximates the behavior of a steady state TCP flow using packets of size s. Most of the long-lived TCP sessions on the internet are Web or FTP transfers that normally use the MTU of the path as the packet size. Therefore for most of the TCP sessions that reach the steady state s is equal to the MTU of the path between the endpoints of the connection. For continuous media applications the situation is different. There are two main reasons why a CM application may not use packets of size equal to the MTU. The first is the delay caused by using big packets. For example a packet of 1400 bytes can carry more than 1.3 seconds of speech coded in G.729. In an voice conversation waiting for 1.3 seconds to send the speech to the other end will ruin the interactivity between the speakers. So small packets are used to keep the end to end delay sufficiently low. The second reason for using small packets is that if too much information is carried in a packet and that packet is lost there can be a strong degradation in the perceived quality so it may be better to send the information in smaller packets so that the loss of some of them can be tolerated (1).

From the discussion above it is clear that for some CM applications the value of s in equation (1) can be smaller than the MTU. Those applications would get less bandwidth than a TCP session using a value of s equal to the MTU if we use equation one to determine their sending rate. This difference can be significant for some applications. So the following question arises : Is this situation fair ?

We now look at the previous definitions of a TCP friendly or Compatible flow.
The concept of TCP Friendliness as defined in [4] is : "We say a flow is TCP-friendly if its arrival rate does not exceed the arrival rate of a conformant TCP connection under the same circumstances ".
In [8] the concept of "TCP-compatible" flow is introduced as "a flow that behaves under congestion like a flow produced by a conformant TCP. A TCP-compatible flow is responsive to congestion notification, and in steady-state it uses no more bandwidth than a conformant TCP running under comparable conditions (drop rate, RTT, MTU, etc.)".

Both definitions consider fair that flows using small packet sizes get less bandwidth than flows using the MTU. We propose to modify the definition as follows : A flow is TCP Compatible if it behaves under congestion like a flow produced by a conformant TCP. A TCP-compatible flow is responsive to congestion notification, and in steady-state it uses no more bandwidth than a conformant TCP using packets of size the MTU and running under comparable conditions (drop rate, RTT, MTU, etc.).

We argue that if a flow is forced to send small packets because of the nature of the application it should get the same bandwidth as a flow sending big packets. If not, we would favor applications sending large packet just because most TCP applications send large packets in steady-state. We also think that this modification to the concept of TCP friendliness will not lead to starvation to TCP flows sending small packets. The new TCP friendly flows will act as traditional Web or FTP TCP flows and therefore would not introduce any additional unfairness of TCP flows sending small packets.

Variable Packet Size Support

One shortcoming of the TFRC protocol is that it only supports fixed packet size flows. Again while that may be the case for many of the TCP applications in steady state it is not for some CM applications. For example in [5] variable packet sizes are used to adapt the sending rate in a video over the internet application. Variable packet size can also be used to reduce the sending rate in voice over IP applications. There is also a growing interest in the research community in the development of variable bitrate speech coders [9] that will produce variable packet size flows.

Therefore a protocol used for CM applications should be able to deal with variable packet sizes flows. We have modified the TFRC protocol so that it can support variable packet sizes flows and at the same time incorporate the new proposed concept of TCP Friendliness.

The first modification is to change the numerator in equation (1) to be the MTU of the path :

$$T = \frac{MTU}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})\,p(1+32p^2)} \qquad (2)$$

In this way we ensure that the flow will get the same amount of bandwidth as a typical TCP flow. We assume that R and t_{RTO} are not greatly affected by the packet size. Therefore the only remaining parameter is *p* the loss even rate.

We compute the loss event rate *p* in the following way :
For each RTT period, we compute the average packet size s_{mean} for the interval by adding the sizes of the received packets and dividing by the total number of packets. Then, for every packet that has been lost we add s_{mean} to a counter that keeps track of loss bytes. If the counter is equal or bigger than the MTU, we act as if a loss of the size of the MTU has just occurred and we use the original TFRC algorithm to see if the loss triggers a lost event. As long as the counter is less than the size of the MTU we do nothing.
In a similar way, for every packet that has been received, we add its size to a counter and if the counter exceeds the size of the MTU, then we act as if a packet of that size had just been received and use the original TFRC algorithm to update the number of packets since the last loss event.
The reasoning behind equal weighting of packet loss independent of the size of the lost packet is that we assume that routers do not take packet sizes into account when they discard packets. Therefore every packet loss is an equally important sign of congestion.

The process is illustrated in figure 1 which shows the received packets of a flow in a RTT period. P2 and P3 are equal to the MTU and the rest of the packets are half the MTU size, we also can see that P4 and P7 where lost. In this case we first calculate the mean value of the packet size for the period which is :

$$s_{mean} = \frac{2*MTU + 4*\left(MTU/2\right)}{6} = \frac{2*MTU}{3} \quad (3)$$

We start counting the received packets and adding their size to a counter $c_{received}$ .We add the size of P1, then when we add P2 and $c_{received}$ is greater than MTU, so we mark a received packet in the original TFRC algorithm and decrement $c_{received}$ by the MTU. We repeat the process until we arrive at P4 which has been lost. We add s_{mean} to the loss counter c_{lost} .Then, we add the size of P5 to $c_{received}$ and mark a received packet in the original TFRC algorithm and decrement $c_{received}$ by the MTU. We process P6 as before and see that P7 was also lost, so we add s_{mean} to the loss counter c_{lost} which is now greater than the MTU so we mark a loss event[1] in the original TFRC algorithm and decrement c_{lost} by the MTU. The final result is shown in figure 2.

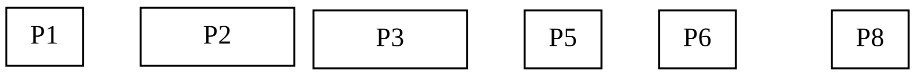

Figure 1 : Example of loss event rate calculation Original Packets

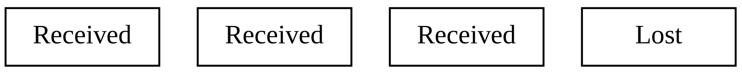

Figure 2 : Example of loss event rate calculation Mapping to TFRC events

This algorithm is backwards compatible to TFRC for constant packet size flows of MTU size because in this case each time a packet is received or lost the counter will equal the MTU and an event will be trigger in the original TFRC algorithm.

If at the end of a RTT period the counters are not zero their value is conserved for the next RTT calculations. We also can make an estimate of s_{mean} for several RTT if there are not sufficient packets on a RTT period.

Simulations of TFRC with Variable packet Size support

To validate our proposed modifications to TFRC we modified the TFRC code in the Network Simulator (ns) [10] and run a number of simulations with different packet sizes flows.

The flows are :

- Constant packet size flows of 1000, 500 and 100 bytes.
- Random packet size uniformly distributed between 250 and 750 bytes.
- Variable packet size controlled by the application (between 250 and 500).

[1] We mark the loss event because it is the first loss in a RTT, if it was not we would not mark it just as TFRC does.

The last flow emulates an application that sends bigger packets in times of congestion. Which can be the case in a voice over IP application which encapsulates more voice frames in each packet in times on congestion to reduce the overhead on the IP-UDP-RTP headers.
These flows share a link with normal TCP-Sack[2] connections using packets of size 1000.The routers use Random Early Detection (RED) queuing. We show the mean sending rate of the modified TFRC and TCP flows over 100 seconds to compare the inter-protocol fairness.

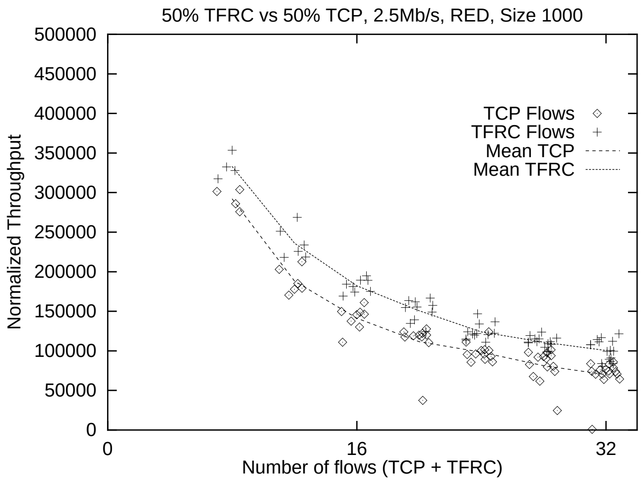

Figure 3 : TCP competing with TFRC (packet size 1000) flows.

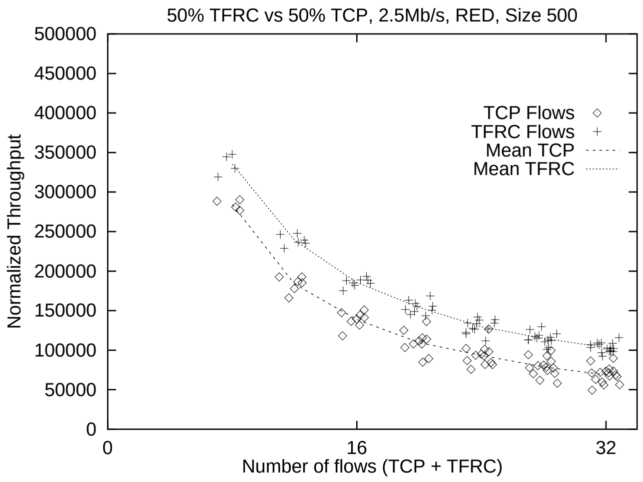

Figure 4 : TCP competing with TFRC (packet size 500) flows.

[2] We use TCP Sack although equation (1) models TCP Reno [2] because equation (1) does not fully cover the impact of TCP timeouts so we minimize them by using TCP-SACK. See Annex B for a further discussion.

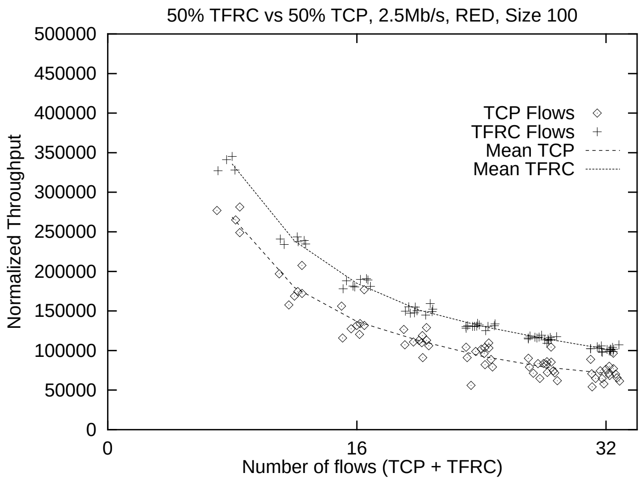

Figure 5 : TCP competing with TFRC (packet size 100) flows.

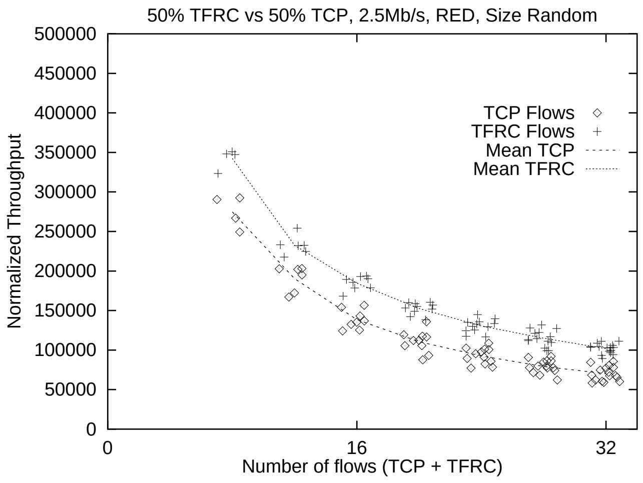

Figure 6 : TCP competing with TFRC (Random Packet Size) flows.

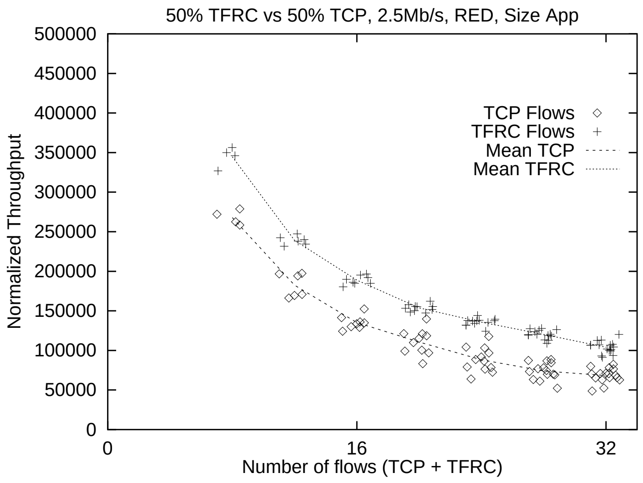

Figure 7 : TCP competing with TFRC (Application Controlled Packet Size) flows.

As can be seen from the graphs in figures 3 to 7, the modified TFRC competes fairly with TCP although it is slightly more aggressive than TCP. We suspected that this was caused by the coarseness of the RTT estimate of TCP as we were using a link delay of 100 ms. We ran some simulations with a link delay of 500 ms to verify that our assumption was true. The results are shown in Annex A of the report.

To ensure that the modified protocol behaves properly, we analyze its response to congestion in various scenarios. In figure 8 we show four TFRC flows that are initiated when we have 4 existing TCP flows and in figure 9 the results are shown for the inverse situation. In both cases the packet size is random (between 250 and 750). Similarly, in figure 10 we show a TFRC flow establishing with other existing TFRC flows.

In the graphs we notice that some variations of the TFRC flows (beginning of figures 8 and 9) seem to be synchronized. This is because the Inter-Packet Space Modulation (ISM)[3] is not currently implemented in the TFRC version available in ns.

We also note that in figure 10 the TFRC flows take longer to establish, this is consistent with the fact that the already active TFRC flows will take longer to react to the congestion introduced by the new flows that the TCP flows in figure 8.

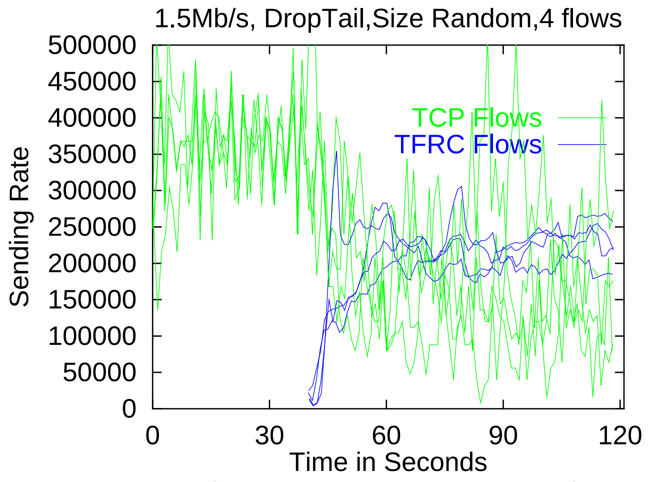

Figure 8 TFRC flows starting with existing TCP flows.

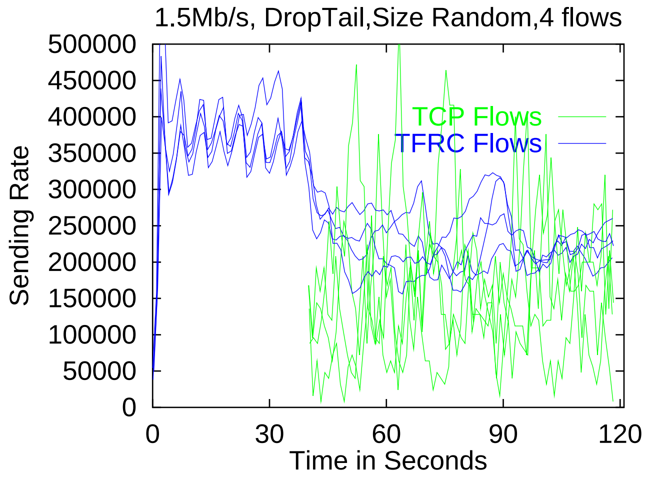

Figure 9 TCP flows starting with existing TFRC flows.

[3] See [3] for a deatiled description of ISM

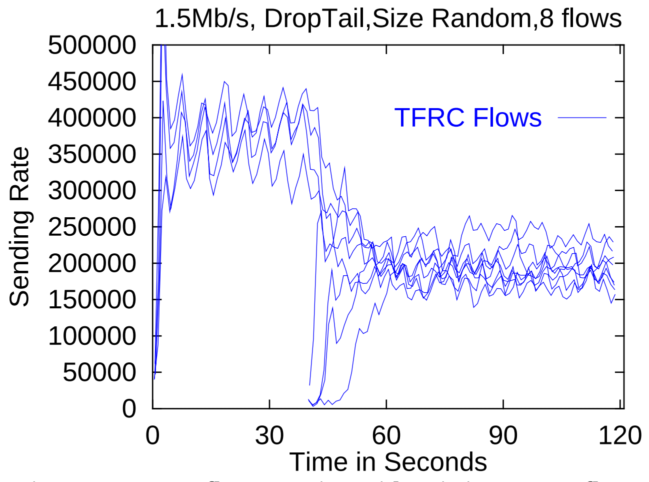

Figure 10 TFRC flows starting with existing TFRC flows.

Analysis of TFRC with real-time Continuous Media

So far the TFRC protocol has been tested assuming that the source always has data ready to send but in many CM applications the situation is slightly different. A codec (for example video codec) is providing video frames periodically based on the rate that we have selected so we may be sending below the allowed rate for some periods of time. We also have to consider that most of these sources have both a maximum and a minimum sending rate so if TFRC is telling the application to send below its minimum sending rate it will probably make sense to stop sending at all or to keep sending at the minimum rate. Here there has to be some interaction between TFRC and the application.

For example consider again a TFRC sender sending voice compressed with G.729 and that in order to reduce bandwidth when the sender is told by TFRC that it has to reduce the sending rate, it does by just putting more voice frames in each IP packet so that the overhead of the headers is reduced and so the bandwidth consumption is lowered (at the cost of an increased delay !!). We allow to send from one to six voice frames in one IP packet and if the sender is already sending six frames per packet and it is asked to reduce the sending rate further it just misbehaves and continues to send the six frames per packet every 60 ms. The results for these simulation are shown in figure 11.

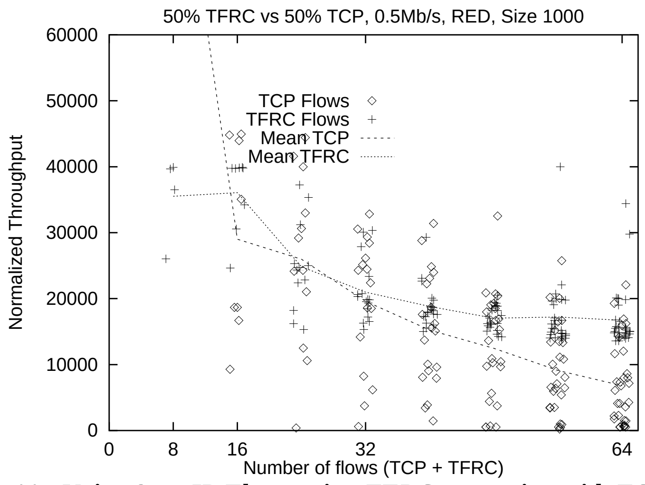

Figure 11 : Voice Over IP Flows using TFRC competing with TCP flows.

In this case the minimum sending rate would be around 13.3 Kbps and the maximum sending rate would be 40 Kbps. The application is fair until the rate approaches the minimum rate of the application. In this case, we would need just to stop the flow and let the user know that the network is not able to provide the service .

In this section we only want to point out that in order to use TFRC in real applications, some further experiments need to be done to evaluate how TFRC behaves with real CM sources.

Conclusion and Open Issues

We have extended previous work on TCP Friendly Congestion Control so that the TFRC protocol can be used with variable packet size flows. The proposed approach has been validated in several simulations and works well in various conditions. We have also revised the concept of TCP friendliness in order to accommodate some multimedia flows. The overall result is an enhanced TFRC protocol that has a wider range of possible applications and therefore more chances of being widely adopted by the internet community. There is though many work that needs to be done in validating the proposed modifications. Extensive testing both on the ns simulator and on the internet needs to be done.

Another important issue is the interaction between the TFRC protocol and the application itself. How can the information that TFRC has on the network conditions we passed to the application so that in can adapt its sending rate while still providing an acceptable service ?

We also consider that equation (1) has only partially captured the effects of retransmission timeouts in TCP and some additional work needs to be done to get a new equation that models TCP behavior better[4].

Acknowledgements

We want to thank J. Widmer for providing his comments and suggestions on how to modify TFRC and for his help during the simulations. We also like to thank J. Padhye and M. Handley for their comments and feedback.

References

[1] S. Floyd, M. Handley, J. Padhye and J. Widmer "Equation Based Congestion Control for Unicast Applications" http://www.aciri.org/tfrc/.

[2] J. Padhye "Towards a Comprehensive Congestion Control for Continuous Media Flows in Best Efforts Networks" PhD Dissertation, University of Massachusetts at Amherst 2000.

[3] J. Widmer "Equation Based Congestion Control" Master's Thesis, University of Mannheim, 2000".

[4] S. Floyd and K. Fall "Promoting the Use of End-toEnd Congestion Control on the Internet" IEEE/ACM Transactions on Networking, Aug 1999.

[5] W.Tan and A.Zakhor. Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol. *IEEE Trans. Multimedia*, Vol. 1, No. 2, pp 172-186, June 1999.

[6] ITU-T Recommendation G.729 "Coding of Speech at 8 kbps Using Conjugate Structure Algebraic Code Excited Linear Prediction (CS-ACLEP) " March 1996.

[7] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson "RTP : A transport Protocol for Real Time Applications" RFC 1889.

[8] B. Braden, Clark, D., Crowcroft, J., Davie, B., S. Deering, Estrin, D., Floyd, S., V. Jacobson, G. Minshall, Partridge, C., L. Peterson, Ramakrishnan, K.K., Shenker, S., Wroclawski, J., Zhang, L., Recommendations on Queue Management and Congestion Avoidance in the Internet, April 1998. RFC 2309 (text), Informational.

[4] See the discussion in Annex A.

[9] A. Das, E. Paksoy and A. Gersho "Multimode and Variable Rate Coding of Speech" in "Speech Coding and Synthesis", Editors W.B. Klejin and K.K. Paliwal, Elsevier 1995.

[10] "The Network Simulator (NS) " http://www-mash.cs.berkeley.edu/ns/ns.html/.

[11] K. Fall and S. Floyd "Simulation-based Comparisons of Tahoe, Reno and Sack TCP " Computer Communication Review V. 26 N. 3 July 1996.

Annex A Simulations with a link delay of 500 ms

Here we present the same simulations as in figures 3 to 5 except for that the link delay is 500ms. This shows that the aggressiveness of TFRC was due to the coarseness in the RTT estimation in TCP.

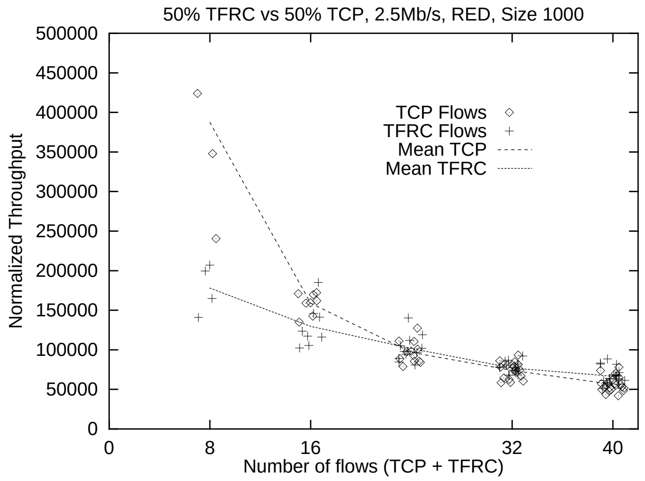

Figure 12 : TCP competing with TFRC (packet size 1000) flows.

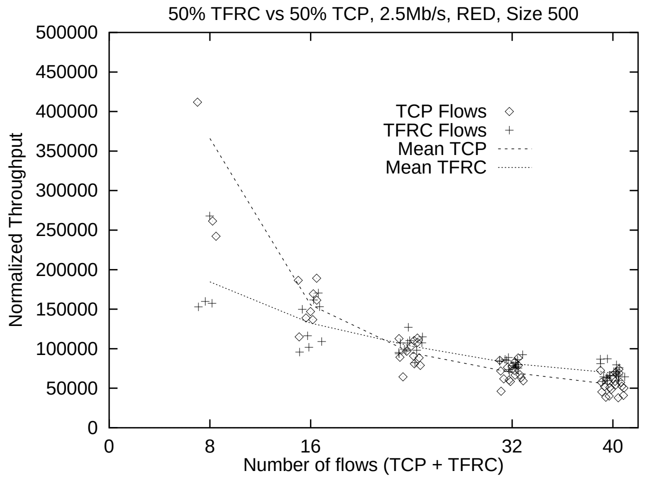

Figure 13 : TCP competing with TFRC (packet size 500) flows.

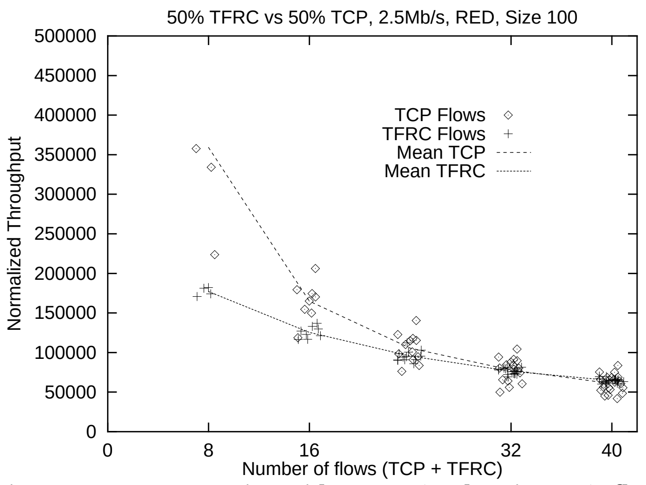

Figure 4 : TCP competing with TFRC (packet size 100) flows.

Annex B

Here we discuss in some detail the shortcomings of equation (1) in modeling the TCP timeout mechanisms. The reader is strongly encouraged to read the chapter 2 of [2] for background information.

The derivation of equation (1) as a model for TCP Reno is presented in the second chapter of [2]. The assumption is that packet losses are correlated within a RTT period so that if a packet is lost, so too are all packets that follow until the end of that RTT period. Suppose that we have a round of w packets *f1….fw* (where w would be the size of the TCP window) and *f1….fk* are acknowledged and *fk+1….fw* are lost. As *f1….fk* have been acknowledged the sender will send packets *s1….sk* in the following round. If the number of packets received is higher than three a Triple Duplicate Ack will occur and the sender will retransmit *fk+1*. In other case a timeout will occur. Based on these scheme, a formula is derived for the probability of a loss triggering a timeout event.

The problem is that this derivation does not take into account that the other packets *fk+2….fw* have also been lost and that for TCP-Reno this would most probably lead to a subsequent retransmission timeout [11]. Instead [2] considers that no more timeouts appear on that round. For example, following [11] if 3 packets are loss in a round and the space between the first and the second loss is zero packets, TCP Reno will timeout. So when $k < W-2$ we will always have a timeout. These timeouts are not predicted by equation (1).

It is now clear that equation (1) only captures some of the TCP timeouts. That is the reason of using TCP Sack in our simulations to minimize the number of timeouts so that (1) still models correctly TCP behavior.