An Extension of the TCP Steady-State Throughput Equation for Parallel Flows and Its Application in MulTFRC

Dragana Damjanovic and Michael Welzl

***Abstract*—In the first part of this paper, we present a simple extension of the well-known TCP steady-state throughput equation that can be used to calculate the throughput of several flows that share an end-to-end path. The value of this extension, which we show to work well with simulations as well as real-life measurements, is its practical applicability. Thus, in the second part of this paper, we present its application in MulTFRC, a TCP-friendly rate control (TFRC)-based congestion control mechanism that is fair to a number of parallel TCP flows while maintaining a smoother sending rate than multiple real TFRC flows do. MulTFRC enables its users to prioritize transfers by controlling the fairness among them in an almost arbitrary fashion.**

***Index Terms*—Computer networks, modeling, protocols.**

I. Introduction

MORE than a decade after its publication, the steady-state throughput equation from [1] remains the most widely used method for calculating the rate of a TCP sender under certain network conditions. However, this model only considers a single TCP flow, and simply multiplying its output with the number of flows under consideration does not work. This limits its utility—for example, it is common for Web browsers and peer-to-peer applications to open connections in parallel.

In this paper, we derive an extension of the equation from [1] to multiple flows. We do this by following the basic approach as in the original paper, but considering a number of senders using an identical path at the same time instead of a single one. End-to-end flows between the same two hosts can sometimes traverse different paths, e.g., when traffic engineering with equal-cost multipath (ECMP) is applied; such situations are not captured by our model.

We then use our new model to turn TCP-friendly rate control (TFRC) [2], a congestion control mechanism that is suitable for multimedia applications because of its smooth sending rate, into *MulTFRC*. MulTFRC is n-TCP-friendly, i.e., as TCP-friendly as n TCP flows, with a faster reaction to congestion and a smoother rate than exhibited by n separate TFRC flows. It can be used for various purposes—e.g., for new application protocols or tunneling protocols that multiplex several application-level streams onto one TCP flow, allowing the composite TCP flow to be given the same overall aggressiveness as it would have had if the application had actually used n separate TCP flows.

MulTFRC could also be used to let users control the importance of traffic within their own traffic mix according to their preferences, including the ability to make it less-than-TCP-friendly $(0 < n < 1)$. This is of interest for so-called "scavenger" traffic that should have a notable influence on other traffic, or for splitting traffic across multiple paths while being as aggressive as one TCP flow if these paths traverse a common bottleneck at some point in the network—a strategy that is currently pursued for multipath TCP in the IETF.[1]

Both the equation and MulTFRC have been presented before [3]. Here, we extend this work by providing a full derivation of the equation and extensive real-life validation results, which we present in Section II, as well as more MulTFRC simulations and real-life tests in Section III. Some of the results from [3] are repeated for the sake of completeness. Related work is discussed in Section IV, and Section V concludes the paper.

II. Model

The equation in [1] provides the sending rate of a TCP flow as a function of the round-trip time (RTT) and "loss events" that a flow experiences, where a loss event is defined as one or more packet loss occurrences during an RTT. For extending this equation to multiple TCPs, a naive approach would be simply to multiply the equation from [1] with n. However, as also discussed in [4], for such a calculation to work, loss-event probability measurements of all n single flows would be needed, i.e., they would have to be observed separately.

This problem is illustrated in Fig. 1, which shows a scenario where three out of four flows experience loss within the same RTT. The measured input for the equation from [1] is just that a loss event occurred—whether one, two, three, or all four flows were affected is not considered by the model. This, however, causes a pronounced difference in the dynamic behavior of the senders, leading to a different cumulative throughput.

Fig. 2 shows how wrong it can be to simply feed the measured cumulative loss-event ratio into n times the TCP steady-state throughput equation. This particular example is based on

Manuscript received August 08, 2010; revised February 22, 2011; accepted March 08, 2011; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Allman. Date of publication April 29, 2011; date of current version December 16, 2011.

D. Damjanovic is with the University of Innsbruck, Innsbruck 6020, Austria (e-mail: dragana.damjanovic@uibk.ac.at).

M. Welzl is with the University of Oslo, Oslo 0316, Norway (e-mail: michawe@ifi.uio.no).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2011.2142008

[1]https://datatracker.ietf.org/wg/mptcp/.

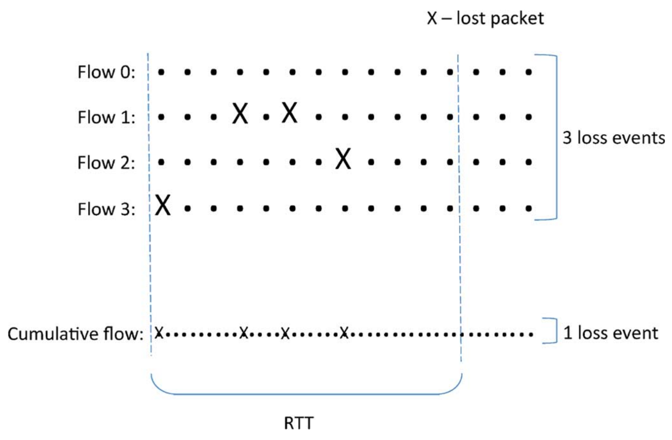


Fig. 1. Per-flow versus cumulative flow loss measure.

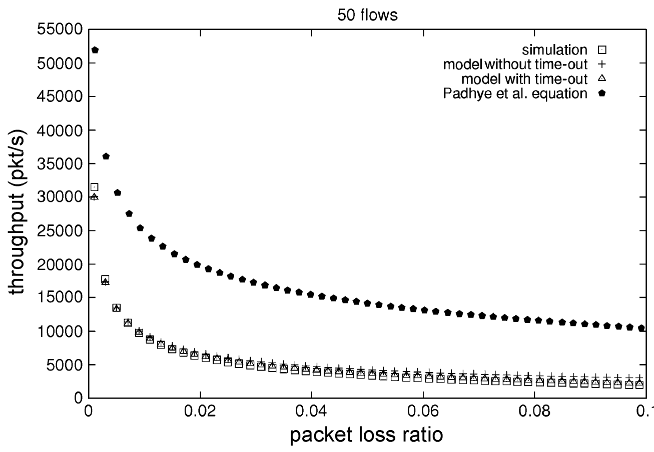


Fig. 2. Our model versus $n*$ the model from [1] (denoted by "Padhye *et al.* equation").

simulations; it is the same as in Fig. 4(a), fully described in Section II-C.

Rather than measuring loss events for each individual flow, from a practical point of view, it is much easier to observe n flows as a single cumulative flow and measure its loss-event rate. Hence, this is what we use as input for our model. In a loss event of the cumulative flow, one or more flows can experience loss, and to estimate this number, we additionally use information about the packet loss rate.

In what follows, we derive an equation that yields the average steady-state throughput of n parallel TCP flows as a function of the loss-event rate, packet loss rate, RTT, TCP retransmission timeout value, and the maximum number of packets acknowledged by a single ACK. Except for n and the packet loss rate, all of these inputs are similar to [1], and so is our approach: Essentially, we repeat the derivation, but assume that we now have n flows instead of one.

Consider n parallel TCP flows $f_1, \ldots, f_n$ starting at time $t = 0$. As in [1], we model TCP's congestion avoidance phase with "rounds," assuming furthermore that the flows are synchronized in terms of rounds (i.e., in a round, all flows send their current window size W_{f} before the next round starts for all flows). While this assumption may seem a bit far-fetched, it is a necessity for keeping the model simple. Since we are dealing with cumulative long-term averages, the exact synchronization behavior between flows is no major concern.

For any given time $t \geq 0$, we define N_t as the number of packets transmitted by all flows in the interval $[0, t]$. Let $B_t := N_t/t$ be the cumulative throughput of all flows in that interval.[2]

Then, we can define the long-term steady-state throughput

$$B := \lim_{t\to\infty} B_t = \lim_{t\to\infty} \frac{N_t}{t}.$$

Let W be the cumulative window size of all flows, and b be the number of packets acknowledged by a received ACK. The TCP protocol reduces its rate in case of congestion, which can be indicated through duplicate acknowledgements and timeouts. TD denotes a "triple duplicate" acknowledgment, i.e., receiving four ACKs with the same sequence number. First, we only consider TD events as loss indications, meaning that the flows stay in the congestion avoidance phase. Timeout loss indications will be incorporated later.

We assume that all flows share the same path and have the same average RTT. They are interleaved (i.e., in a round, packet 1 belongs to f_1, packet 2 to f_2, etc.). This assumption is hard to justify because the actual pattern of packets will vary depending on the dynamic behavior (e.g., when the two flows were started). Such fine-grain dynamism is not captured by our model. However, the packet order only influences the decision as to which flow a lost packet belongs. We had to decide for a specific pattern for simplicity, and the effect of taking a wrong assumption can be expected to be minimal because only one to two packets are lost in a burst [5] or packets are even dropped randomly in case of a RED queue. Whenever a flow f experiences a TD loss indication, it halves its window size W_{f}. We observe TD loss indications of the cumulative flow. In such a TD event, one or more individual flows experience loss. For n parallel flows, we define a TD-period (TDP) as a period between two consecutive TD loss indications of the cumulative flow. p_{e} is defined as the probability that a packet is the first packet lost in a loss event of the cumulative flow.

We do not make the same assumption as in [1]: If a packet is lost, all consecutive packets belonging to the same window are lost as well. We believe that this decision is well justified: From [5], it is known that the number of packets that are usually lost in a row in the Internet is limited ("single packet losses account for the large majority of the bursts, around 83%, and double losses occupy 12% of the bursts"). Since we assume packets from flows to be interleaved in a round-robin fashion, multiple consecutive packet drops would affect multiple flows and have a more severe (potentially erroneous) impact on our model than it has on the original model in [1].

For the ith TD-period, Y_i is the number of packets sent in the period, A_i is the duration of the period, and X_i is the number of rounds in the period. It can be shown [1] that

$$B = \frac{E[Y]}{E[A]}. \tag{1}$$

[2]This is similar to [1], where it is stated that "B_t is the number of packets sent per unit of time regardless of their eventual fate (i.e., whether they are received or not). Thus, B_t represents the throughput of the connection, rather than its goodput." The terms "throughput" and "goodput" are sometimes ambiguous. In fact, both the equation in [1] and ours describe the sending rate of a (number of) TCP sender(s) and not what arrives at the receiver. We maintain the "throughput" terminology for consistency with [1].

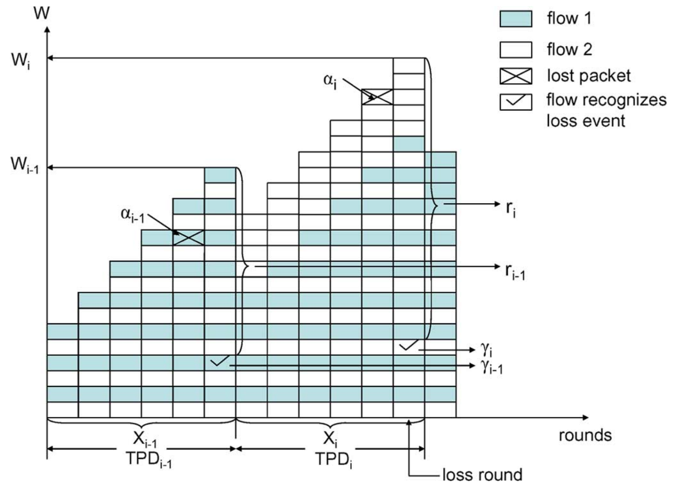


Fig. 3. Evolution of the cumulative send window W of two flows; two triple duplicate periods (TDPs) are shown. "1, 2, 3, 4" insdicates the sequence in which packets are transmitted.

As in [1], if we consider the duration of each round to be a random variable independent of the window size with the average value RTT, and $E[X]$ is the average number of rounds in a TD-period, we have $A_i = \sum_{j=1}^{X_i} d_{ij}$, where d_{ij} is the duration of the jth round in the ith TD-period and

$$E[A] = E[X]\mathrm{RTT}. \tag{2}$$

To derive B, we need to take a closer look at how the evolution of the window size of each flow (W_f) and the time between two loss events of an individual flow (A_f) influence the development of the cumulative window size (W).

Since we derive the model by looking at the TDPs of the cumulative flow, in a loss event of the cumulative flow, more than one flow can experience loss (each flow does not have to experience loss at the end of each TD-period). Let j_i be the number of individual flows that experiences loss at the end of the ith TD-period. Assuming that loss is identically distributed over all flows, the probability that a flow is hit in the ith TD-period is j_i/n. The probability that the time between two loss events of a flow (A_f) is k TD-periods $(k = 1, 2, \ldots)$ is equal to the probability that the flow did not lose a packet in $(k-1)$ consecutive TD-periods, but loses a packet in the kth TD-period

$$P[\text{loss in the } k\text{th TDP}] = \frac{j_i}{n} \prod_{l=1}^{k-1} \left(1 - \frac{j_{(i-l)}}{n}\right). \tag{3}$$

If j is the mean value of the number of flows hit in a round, we have

$$P\left[A_\mathrm{f} = kE[A]\right] = \frac{j}{n}\left(1 - \frac{j}{n}\right)^{k-1}. \tag{4}$$

The mean value of A_f, the time between two loss events of an individual flow, is

$$E[A_\mathrm{f}] = \sum_{k=1}^{\infty} \left(\frac{j}{n}\left(1 - \frac{j}{n}\right)^{k-1} kE[A]\right) = \left(\frac{nE[A]}{j}\right). \tag{5}$$

For deriving $E[Y]$, we will examine the evolution of the cumulative window (W) on the sender side, shown in Fig. 3. In each round, W is incremented by n/b, hence the number of packets sent per round is incremented by n every b rounds. α_i denotes the sequence number of the first packet lost in TDP_i (for simplicity, we assume sequence numbers to begin at 1 for every TD-period). After receiving a triple duplicate acknowledgment, a flow recognizes that loss event (receiving the ACK for packet γ_i). We consider that a TD-period ends when a flow recognizes a loss event. This can happen in the same round or in the next one; the round that γ_i belongs to is called the "loss round." The total number of packets sent in X_i rounds in TDP_i is $Y_i = \gamma_i$, hence

$$E[Y] = E[\gamma]. \tag{6}$$

The probability that $\gamma_i = k$ is equal to the probability that $k-1$ packets are not loss indications, and the ACK of the kth packet triggers the fast retransmission

$$P[\gamma_i = k] = (1 - p_\mathrm{e})^{k-1} p_\mathrm{e}, \qquad k = 1, 2, \ldots \tag{7}$$

and the mean value of γ is

$$E[\gamma] = \sum_{k=1}^{\infty} (1 - p_\mathrm{e})^{k-1} p_\mathrm{e} k = \frac{1}{p_\mathrm{e}}. \tag{8}$$

In the loss round of each TD-period, one or more flows experience loss. For the ith TD-period, let flows $\{m^e\}$, $e = 1 \ldots j_i$, (subset of n flows) be the j_i flows experiencing loss at the end of the period. We will observe the evolution of the window size of an individual flow (flow m^e). Flow m^e does not experience loss in every TD-period, so the TD-periods in which flow m^e experiences loss are a subset ($\{i_s\}$, $s = 1, 2 \ldots$) of TD-periods of the cumulative flow. If $W_{\mathrm{f}_{m^e}{}_{i_s}}$ is the window size of flow m at the end of the (i_s)th TD-period, and $X_{\mathrm{f}_{m^e}{}_{i_s}}$ is the number of rounds from the end of $\mathrm{TDP}_{i_{s-1}}$ till the end of TDP_{i_s}, during $X_{\mathrm{f}_{m^e}{}_{i_s}}$ rounds the window size of flow m^e increases by $X_{\mathrm{f}_{m^e}{}_{i_s}}/b$ (as in [1]) and we have

$$W_{\mathrm{f}_{(m^e)_{i_s}}} = \frac{W_{\mathrm{f}_{(m^e)_{i_{s-1}}}}}{2} + \frac{X_{\mathrm{f}_{(m^e)_{i_s}}}}{b}. \tag{9}$$

The number of packets sent in a TD-period is the number of packets sent between two loss events of the cumulative flow. For the ith TD-period, this includes packets sent in the last round of the $(i-1)$th TD-period, starting from the (γ_{i-1})th packet till the end of the window (r_{i-1} packets, as shown in Fig. 3) and packets sent in next X_i rounds till the (γ_i)th packet. If flows $\{m^e\}$, $e = 1 \ldots j_{i-1}$ are the flows that experience loss in the $(i-1)$th TD-period and $W_{\mathrm{f}_{(m^e)_{i-1}}}$, $e = 1 \ldots j_{i-1}$ are the windows of these flows at the end of the $(i-1)$th TD-period, the window size of the cumulative flow at the beginning of the ith TD-period is $W_{i-1} - \sum_{e=1}^{j_{i-1}} (W_{\mathrm{f}_{(m^e)_{i-1}}}/2) + (n - j_{i-1})$ (the j_{i-1} flows reduce their windows, and the remaining $(n - j_{i-1})$ flows increase their window size by 1). Every b rounds, the window size of the cumulative flow is increased by n, and we have

$$Y_i = r_{i-1} + \sum_{k=0}^{X_i/b-1} \left(W_{i-1} - \sum_{e=1}^{j_{i-1}} \frac{W_{\mathrm{f}_{(m^e)_{i-1}}}}{2} + (n - j_{i-1}) + nk\right) b - r_i. \tag{10}$$

Assuming that $\{X_{\mathrm{f}_{m^e}{}_{i_j}}\}$ and $\{W_{\mathrm{f}_{m^e}{}_{i_j}}\}$ are mutually independent sequences of i.i.d. random variables (as in [1]), from (9) we have

$$E\left[W_{\mathrm{f}_{m^e}}\right] = \frac{2}{b} E\left[X_{\mathrm{f}_{m^e}}\right]. \tag{11}$$

Taking into account the assumption that a loss occurs identically distributed over all flows we can say that $E[X_{\mathrm{f}_{m^e}}]$ and $E[W_{\mathrm{f}_{m^e}}]$ are equal for all flows, and from now on we denote them by $E[X_\mathrm{f}]$ and $E[W_\mathrm{f}]$. From (2) and (5), we have

$$E[X_\mathrm{f}] = \frac{nE[X]}{j}. \tag{12}$$

We assume that at the end of a TD-period j, flows experiencing loss in that TDP have the window size $E[W_\mathrm{f}]$, and other j flows that experience loss in the previous loss event have the window size $\frac{E[W_\mathrm{f}]}{2} + \frac{E[X]}{b}$, etc. The mean value of the window size of the cumulative flow is

$$E[W] = jE[W_\mathrm{f}] + \sum_{k=1}^{\frac{n}{j}-1} j \left(\frac{E[W_\mathrm{f}]}{2} + \frac{kE[X]}{b} \right). \tag{13}$$

From (11)–(13), we have

$$E[W] = \frac{nE[X]}{2b} + \frac{3n^2E[X]}{2bj}. \tag{14}$$

From (10), assuming that a loss occurs independently distributed over the size of the cumulative window in a loss round, hence $E[r] = \frac{E[W]}{2}$, we have

$$E[Y] = \left(E[W] - j\frac{E[W_\mathrm{f}]}{2} + (n-j) \right) E[X] + \frac{nE[X]^2}{2b} - \frac{nE[X]}{2} \tag{15}$$

and including (6), (8), (11), (12), and (14)

$$\frac{1}{p_\mathrm{e}} = \frac{3n^2E[X]^2}{2bj} + \frac{nE[X]}{2} - jE[X]. \tag{16}$$

Solving this equation for $E[X]$, we get (17), shown at the bottom of the page, and including (14), we get (18), shown at the bottom of the page. From (1), (2), (6), (8), and (17), we have (19), shown at the bottom of the page.

As already stated, according to [5], loss is usually not clustered, and so the probability that, in a loss event, more than one packet of a flow is lost is low. In this case, we can approximate j with the average number of packets lost in a loss event, and for a higher loss probability, the flow rate is more likely to be controlled by timeouts and less by fast retransmissions. Thus, this assumption could introduce just a minor error. Therefore, we take the approximation $j = p_\mathrm{r}/p_\mathrm{e}$, where p_r is the probability that a packet (belonging to any flow) is lost. Since j must be less than n, we have $j = \min(n, p_\mathrm{r}/p_\mathrm{e})$.

A. Model With Timeouts

We now extend the equation to include timeouts (TO). A loss event experienced by a flow can be a triple-duplicate loss indication or a timeout loss indication, so in a TD-period, a flow can be in the slow start phase or in the congestion avoidance phase. We denote with Y_{TDP_i} the number of packets sent by flows that are in the congestion avoidance phase in the ith TD-period, and with R_{TO_i} the number of packets sent by flows that are in the slow start phase in the ith TD-period. The long term steady-state throughput B_ext is

$$\begin{aligned} B_\mathrm{ext} &= \frac{E[Y_\mathrm{TDP}] + E[R_\mathrm{TO}]}{E[A]} \\ &= E[B_\mathrm{TDP}] + \frac{E[R_\mathrm{TO}]}{E[A]}. \end{aligned} \tag{20}$$

$E[B_\mathrm{TDP}]$ denotes the throughput of flows that are not in the slow start phase.

We assume that during a TD-period most of the flows are in the congestion avoidance phase and that the number of flows in the slow start phase is considerably smaller. We note that this may be incorrect when loss is very large, but then, even if considering that most of the flows were in the congestion avoidance phase, they would significantly reduce their window size, and

$$E[X] = \frac{2j^2p_\mathrm{e}b - np_\mathrm{e}bj + \sqrt{n^2p_\mathrm{e}^2b^2j^2 - 4np_\mathrm{e}^2b^2j^3 + 4j^4p_\mathrm{e}^2b^2 + 24n^2p_\mathrm{e}bj}}{6n^2p_\mathrm{e}} \tag{17}$$

$$\begin{aligned} E[W] = {} & \frac{2j^2p_\mathrm{e}b - np_\mathrm{e}bj + \sqrt{n^2p_\mathrm{e}^2b^2j^2 - 4np_\mathrm{e}^2b^2j^3 + 4j^4p_\mathrm{e}^2b^2 + 24n^2p_\mathrm{e}bj}}{4bp_\mathrm{e}j} \\ & \times \frac{2j^2p_\mathrm{e}b - np_\mathrm{e}bj + \sqrt{n^2p_\mathrm{e}^2b^2j^2 - 4np_\mathrm{e}^2b^2j^3 + 4j^4p_\mathrm{e}^2b^2 + 24n^2p_\mathrm{e}bj}}{12bnp_\mathrm{e}} \end{aligned} \tag{18}$$

$$B = \frac{1}{\mathrm{RTT}} \frac{6n^2}{2j^2p_\mathrm{e}b - np_\mathrm{e}bj + \sqrt{n^2p_\mathrm{e}^2b^2j^2 - 4np_\mathrm{e}^2b^2j^3 + 4j^4p_\mathrm{e}^2b^2 + 24n^2p_\mathrm{e}bj}} \tag{19}$$

the small total window of the cumulative flow would render the error introduced by this wrong assumption negligible. Taking this assumption, we have

$$E[B_{\mathrm{TDP}}] = B\frac{n - E[n\mathrm{TO}]}{n} \tag{21}$$

where B is defined in (19) and $E[n\mathrm{TO}]$ is the average number of flows that are in the slow start phase. The second part of (20) $(E[R_{\mathrm{TO}}]/(E[X]\mathrm{RTT}))$ is the throughput of flows that are in slow start. It equals $E[n\mathrm{TO}](E[R]/E[Z^{\mathrm{TO}}])$, where $E[R]$, following the notation from [1], is the average number of packets sent by one flow during a timeout sequence of average duration $E[Z^{\mathrm{TO}}]$. $E[R]$ and $E[Z^{\mathrm{TO}}]$ are calculated in the same way as in [1], and they are

$$E[R] = \frac{1}{(1-p_{\mathrm{e}})} \tag{22}$$

$$E[Z^{\mathrm{TO}}] = T * \frac{f(p_{\mathrm{e}})}{1-p_{\mathrm{e}}} \tag{23}$$

where $f(p) = 1 + p_{\mathrm{e}} + 2p_{\mathrm{e}}^2 + 4p_{\mathrm{e}}^3 + 8p_{\mathrm{e}}^4 + 16p_{\mathrm{e}}^5 + 32p_{\mathrm{e}}^6$, and T denotes the initial period of time (in a TO phase) after which the sender retransmits unacknowledged packets.

$E[n\mathrm{TO}]$ also includes flows in the slow start phase due to receiving a timeout loss indication not only in the current loss round, but also in the previous loss rounds, and it is equal to $E[n\mathrm{TO}'](E[Z^{\mathrm{TO}}]/(E[X]\mathrm{RTT}))$, where $E[n\mathrm{TO}']$ is the number of flows experiencing timeouts in a loss round.

To derive $E[n\mathrm{TO}']$, let $p\mathrm{Lost}_i$ be the number of packets lost in TDP_i, and let l_i flows (flows $\{m^e\}$, $e = 1\ldots l_i$) experience timeouts at the end of TDP_i. If $W_{\mathrm{f}_{(m^e)_i}}$, $e = 1\ldots l_i$ are congestion window sizes of these flows at the end of TDP_i, we have $p\mathrm{Lost}_i \geq \sum_{e=1}^{l_i} W_{\mathrm{f}_{(m^e)_i}}$. Assuming that $\{p\mathrm{Lost}_i\}$ and $\{W_{\mathrm{f}_{(m^e)_i}}\}$ are sequences of mutually independent random variables, and taking the average window size of a flow to be $E[W]/n$ and l to be the average number of flows experiencing timeouts in a loss event, we have: $p\mathrm{Lost} \geq l(E[W]/n)$. Since flows experiencing a triple-duplicate loss indication lose far fewer packets than flows experiencing a timeout loss indication, we can approximate that the number of flows experiencing a timeout in a loss event is

$$E[n\mathrm{TO}'] = \min\left(\frac{n\,p\mathrm{Lost}}{E[W]}, n\right).$$

The average number of packets lost in a loss event is $p\mathrm{Lost} = p_{\mathrm{r}}/p_{\mathrm{e}}$. The congestion window size of the cumulative flow $(E[W])$ is given in (18), and the number of flows in the slow start phase in a TD-period is

$$E[n\mathrm{TO}] = \min\left(\frac{n\frac{p_{\mathrm{e}}}{p_{\mathrm{r}}}}{E[W]}, n\right)\frac{E[Z^{\mathrm{TO}}]}{E[X]\mathrm{RTT}}.$$

Using expressions for $E[W]$ from (14) and for $E[Z^{\mathrm{TO}}]$ from (23), we have

$$E[n\mathrm{TO}] = \min\left(\frac{n\frac{p_{\mathrm{e}}}{p_{\mathrm{r}}}}{\frac{nE[X]}{2b} + \frac{3n^2E[X]}{2bj}}, n\right)\frac{f(p_{\mathrm{e}})T}{(1-p_{\mathrm{e}})E[X]\mathrm{RTT}}. \tag{24}$$

Finally, the steady-state throughput of n parallel flows can be calculated as

$$B = \frac{n - E[n\mathrm{TO}]}{n\,p_{\mathrm{e}}\,E[X]\,\mathrm{RTT}} + \frac{E[n\mathrm{TO}]}{f(p_{\mathrm{e}})T} \tag{25}$$

where $E[n\mathrm{TO}]$ is given in (24) and $E[X]$ is defined in (17).

B. Algorithm

Here, we give the equation in the form of an algorithm, where x, j_af, a, z, and q are temporary floating point variables, and the input parameters are the following:

- n: the number of parallel TCP flows;
- RTT; the round-trip time in seconds;
- b: the maximum number of packets acknowledged by a single TCP acknowledgement;
- p_e: the loss event rate;
- j: the number of packets lost in a loss event (as we have already stated, this can also be approximated as $\min(n, p_{\mathrm{r}}/p_{\mathrm{e}})$, where p_{r} is the probability that a packet (belonging to any flow) is lost);
- t_RTO: the TCP retransmission timeout value in seconds.

Algorithm 1: Throughput of n parallel flows [packets/s]

```
$j\_af = \min(\max(j,1), n)$;
    $a = p\_e*b*j\_af*\left(24*n^2+p\_e*b*af*(n-2*j\_af)^2\right)$;
    $x = (j\_af*p\_e*b*(2*j\_af-n)+\mathrm{sqrt}(a))/(6*n^2*p\_e)$;
    $z = t\_\mathrm{RTO}*(1+32*p\_e^2)/(1-p\_e)$;
    $q = \min\left(2*j*b*z/\left(\mathrm{RTT}*(1+3*N/j)*x^2\right),\right.$
            $\left.n*z/(x*\mathrm{RTT}), n\right)$;

return $((1-q/n)/(p_e*x*\mathrm{RTT}) + q/(z*(1-p\_e)))$;
```

C. Validation With Simulations

We validated the accuracy of the TCP model with ns-2 ver. 2.31, using the "dumbbell" topology to investigate the behavior of a set of flows sharing the same link. The access links had a bandwidth of 10 Gb/s and a delay of 1 ms, and characteristics of the shared link were changed and will be specified in the following text. The packet size for all flows was 1040 B, which includes 40 B for the TCP/IP header (the same value was used in [2]; we chose it for consistency and easier comparison of MulTFRC and TFRC).

A number of parallel TCP flows were run, and the throughput, the RTT, and the loss-event rate experienced by these flows was measured. The throughput was measured by looking at the number of packets leaving the sender, irrespective of their sequence number. We use the measured average loss-event rate, packet loss probability (to approximate j in our model), and RTT as input parameters to calculate the throughput using our equation with and without the timeout phase.

First, we will present a scenario with a random loss model on the shared link (each packet is lost with the same probability). This is meant as a starting point. Since the loss behavior is perfectly in line with the assumptions that we took in our model, it would be a very bad sign if the model would already perform poorly in this setup.

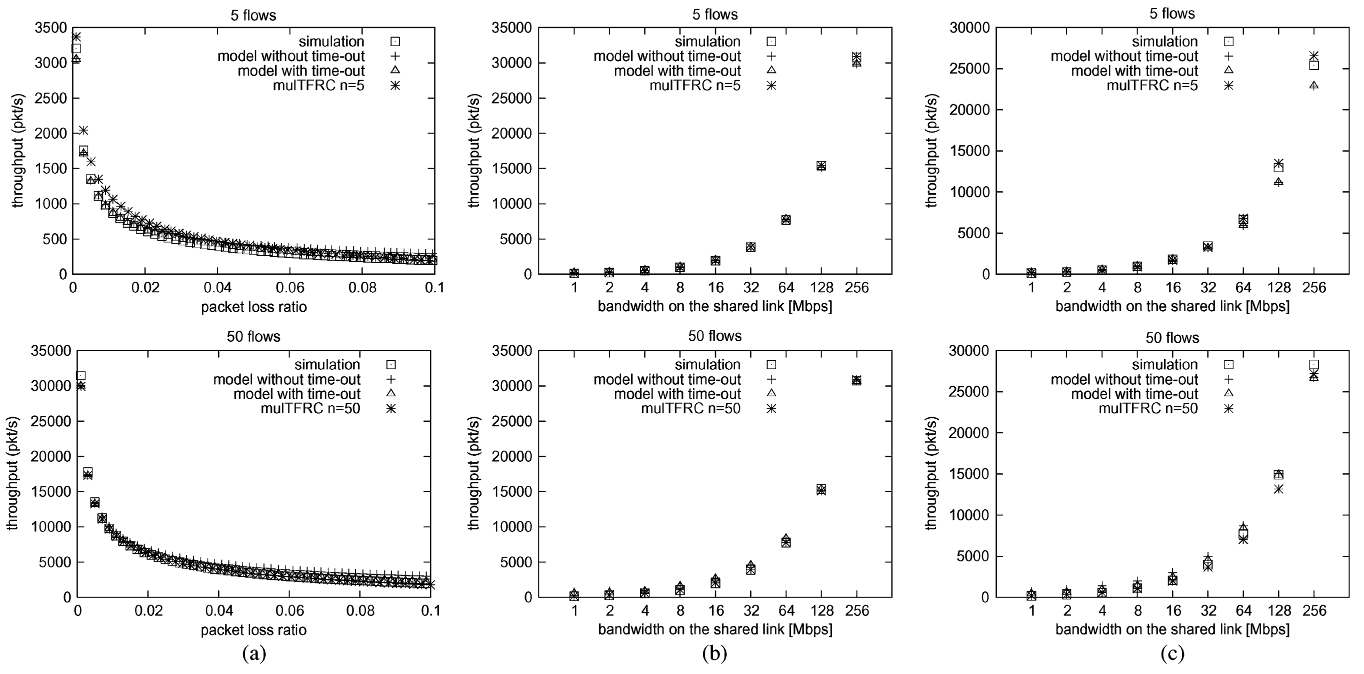


Fig. 4. Comparison of the throughput of TCP flows, the model, and the throughput of MulTFRC. (a) Random loss. (b) No artificial loss, DropTail. (c) No artificial loss, RED.

Each simulation was run for 460 s. The first 15 s of all simulations were not used for throughput and loss-event rate measurements to eliminate the influence of the startup phase because this phase is not included in our model, where slow start only occurs after a timeout has happened. We have cut at least the 10 first seconds from all our measurements. With an initial window of four packets, slow start typically terminates within 5 RTTs for the typical value of 64 kB, and the congestion window grows to an unrealistically large value of 131 072 packets within 16 RTTs. With an RTT as large as 200 ms, slow start still terminates after 3.2 s, and hence cutting 10 s or more is definitely enough to eliminate this phase, including connection setup. Since we generally stopped measurements before the end of simulations or tests, the connection teardown phase is also not included in our measurement data.

The delay of the shared link was 30 ms. The capacity of the shared link was large enough for loss to be produced by the applied loss model only (1 Gb/s). For completeness, we note that a DropTail queue was used on the shared link, although simulations with different queuing mechanisms showed the same results. The amount of loss that is generated by the loss model on the shared link is the parameter that we varied.

The results for 5 and 50 flows are shown in Fig. 4(a) (which also shows the throughput of MulTFRC; this will be discussed in Section III). For a loss rate smaller than 2% (which is to be expected on the Internet today), our equation does not produce an error greater than 3%, except for a very low loss rate (<0.01%). In that case, our equation underestimates the measured throughput, but the error is never greater than 5%.

To remind the reader, we approximate j, the number of individual flows that experience loss events in a loss event of the cumulative flow, by dividing the packet-loss ratio with the loss-event ratio. Even though, in the simulations with this low loss rate (<0.01%), the difference between the approximation and the measured j (the average number of flows experiencing loss in a loss event of the cumulative flow, obtained by looking at each flow separately) is quite small [e.g., 0.05% for the lowest loss rate of five TCP flows shown in Fig. 4(a)], this difference causes the mismatch between our model and simulations. For the described example, the equation calculated using the approximation shows an error of 4.7% and, using the measured value for j, the difference is just 2%. Measuring j would, however, require each flow to be observed separately, and this is what we wanted to avoid. We believe that it is unnecessary because, as the loss rate grows, the rate decreases, and the error produced by the j approximation becomes smaller.

In the second part of our simulations, we let packet loss be produced just by the observed flows alone. The bottleneck delay was 30 ms, and the bottleneck capacity was changed to cause a varying amount of loss. Each simulation was run for 600 s, and first 50 s were not used for throughput measurements to eliminate the influence of the startup phase. First, we will observe the simulations with a DropTail queue on the shared link. The queue length was set to the bandwidth × delay product. The results are shown in Fig. 4(b). For the simulations with a higher capacity (256 and 128 Mb/s) of the shared link, the difference between the simulation and our equation was very small, even less than 1%, except for five flows where the error is around 3%. In this case, the error is again due to the j approximation (the same as in the previous simulations with a low loss rate), and it becomes negligible as the number of flows increases.

Fig. 4(c) shows results when a RED queue was used on the shared link. We allowed the algorithm implemented in ns-2 to automatically configure the RED parameters ($q\_weight = -1$, $\text{thresh} = 0$, $\text{maxthresh} = 0$ were set for automatic configuration, and the "gentle" mode was activated). As in the Drop-

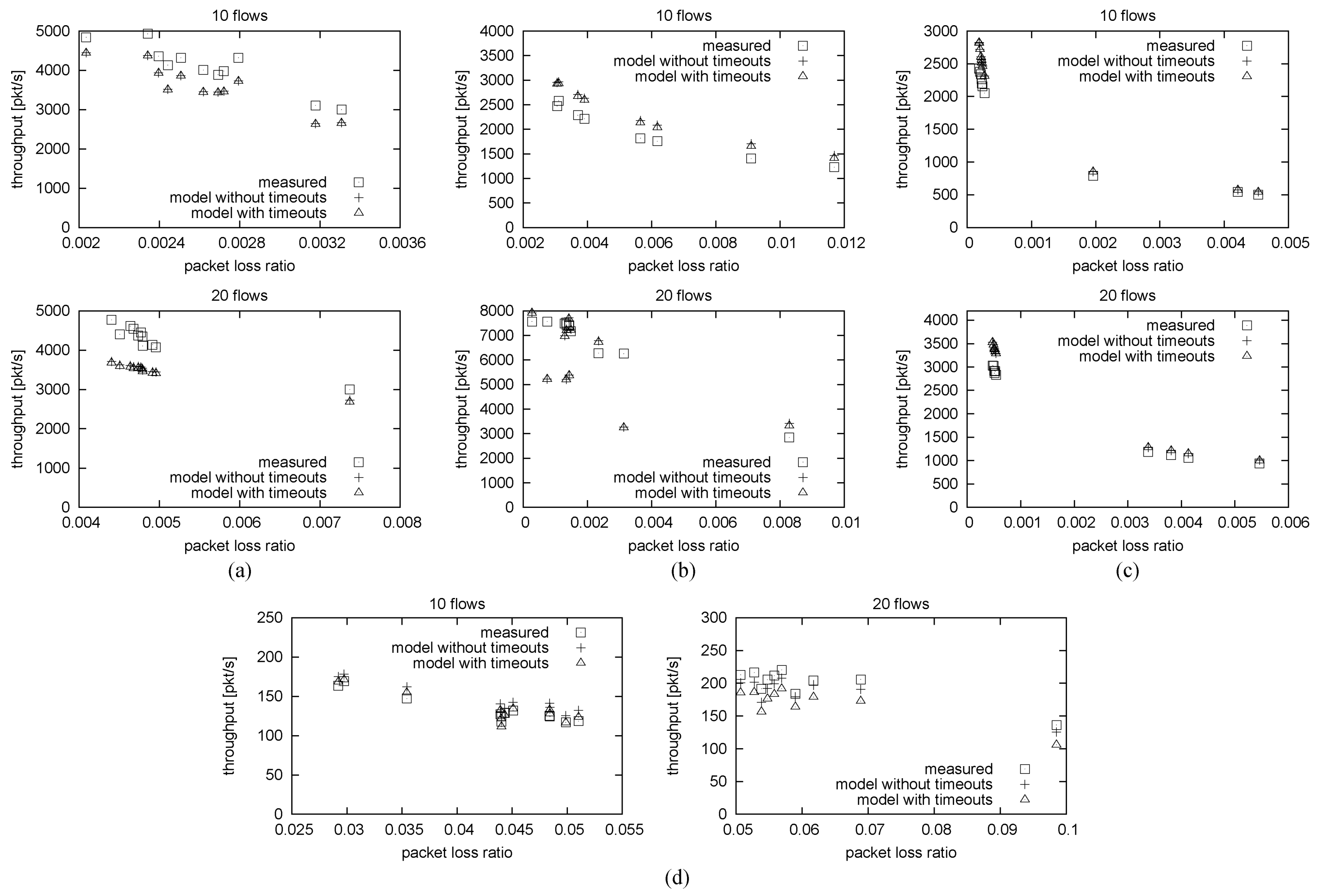


Fig. 5. Model: measurements Innsbruck—PlanetLab nodes. (a) France. (b) Portugal. (c) Korea. (d) Uruguay.

Tail case, the model matches the measured TCP throughput very well for five flows, except for slight error that is introduced by the j approximation for very low loss rates (large capacities).

To investigate the opposite extreme case to random loss, we attempted to validate the accuracy of our model in the presence of bursts. We used a loss model that always drops a certain number of consecutive packets and varied this number. Some shortcomings of the ns-2 simulator were seen here, i.e., there was no multiplexing between flows. We tried (e.g., using RED and the “overhead_” ns-2 parameter [6]), but we could not eliminate this problem. After a few RTTs, packets from the same flow always ended up being sent back to back. Therefore, the dropped packets in a loss event always belonged to the same flow, which is at odds with our model’s assumption of interleaved flows, causing it to underestimate the measured throughput. Since such an extremely synchronized behavior is not to be expected in the Internet, we decided to continue our validation with real-life measurements at this point.

D. Real-Life Measurements

The throughput of TCP flows traversing a path between a host in Innsbruck, Austria, (138.232.65.6, Linux kernel ver. 2.6.30) connected to the University of Innsbruck’s LAN with a 100-Mb/s network card and a number of PlanetLab[3] hosts was measured. We also obtained the average RTT and the loss-event rate for each measurement and used them to verify our model. The comparison between the measured throughput and the throughput estimated by our equation is shown in Fig. 5.

All hosts ran PlanetLab software (kernel ver. *2.6.22.19-vs2.3.0.34.39.planetlab*), TCP window scaling was turned on with an advertised window scaling value of 7, and SACK was enabled. For each measurement, the startup phase of TCP was omitted (the first 20 s were not used). The observed time period was long enough to capture the steady-state behavior of TCP. All measurements were taken between the October 1–9, 2009. We ran 10 and 20 TCP flows and measured the throughput using Web100[4] ver. 2.5.25.

The first receiver was at Universite de Technologie de Troyes, Troyes, France (planetlab2.utt.fr—194.254.215.12). As Fig. 5(a) shows, the model underestimates the throughput by about 10% for all measurements of 10 flows, and between 10% and 20% for 20 flows. A closer look at the data showed that an unusually large number of consecutive losses in a loss event (even greater than 15) causes this mismatch.

The next host was at Universidade do Algarve, Faro, Portugal (planetlab1.fct.ualg.pt—193.136.227.163). For the measurements of 10 flows and a part of the measurements of 20 flows, our equation overestimates the throughput by around 10%. There are some measurements of 20 flows for which our equation gives a poor result. This is because the number of losses

[3]http://www.planet-lab.org/.

[4]http://www.web100.org/.

in a loss event (our j approximation) reaches extremely large values of even 17 for these measurements, while the average number of loss events of individual flows in a loss event of cumulative flow was only around two. Since this was only the case for a few consecutive measurements, we assume that it was caused by an unknown temporary problem on the path, e.g., a route change.

We also measured the throughput of TCP connections between Innsbruck and a host at the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea (csplanetlab2.kaist.ac.kr—143.248.139.55). While it matches the measurements quite well, our equation very slightly overestimates the throughput in all samples, as shown in Fig. 5(c).

Fig. 5(d) shows the results of measurements between Innsbruck and a host at Facultad de Ingenieria, Universidad de la Republica, Montevideo, Uruguay (planetlab-1.fing.edu.uy—164.73.47.242). Here, our equation approximates TCP's real behavior quite well. The measurements for 10 flows are very slightly overestimated by our equation, with a difference of less than 4%. On the other hand, the throughput of 20 flows is slightly underestimated (the error is around 15%). This is due to the j approximation error discussed in the previous section.

Equipped with an equation that appears to capture the behavior of multiple parallel TCP flows reasonably well and that is easy to implement, we now turn our attention to make TFRC emulate the congestion control behavior of n TCPs instead of only one.

III. MulTFRC

Since the MulTFRC protocol is based on TFRC, many input parameters for the rate calculation are measured in the same way (RTT, T, and b). The loss-event probability is measured for the cumulative flow. It is obtained as in TFRC, counting just one loss occurrence per RTT. To obtain a smooth rate, the TFRC protocol uses the weighted average of the last k loss intervals. In [2] and [7], k is set to 8; we use the same setting for MulTFRC.

In Section II the number of packets lost in a loss event is calculated using the approximation $j = p_{\mathrm{r}}/p_{\mathrm{e}}$ (the real loss probability divided by the loss event probability). Since it is possible to precisely count the number of lost packets in a loss event in the MulTFRC protocol, we use this as an input for j instead. For this measurement, TFRC's method for discounting samples of the loss-event probability was also applied.

Our equation assumes that each lost packet belongs to a different flow, which is rarely true in reality. As Fig. 7 (to be discussed in detail later) shows, a MulTFRC flow with a large n (even for $n = 100$) is less affected by this assumption. To improve the performance of MulTFRC with small values of n, we incorporate the possibility that more than one packet belongs to the same flow by assuming that a packet belongs to any of the n flows with the same probability $(1/n)$. The probability that a flow is not affected in a loss event is equal to the probability that all j lost packets belong to the other $n-1$ flows: $((n-1)/n)^j$. The probability that the flow is affected is $1-(1-1/n)^j$. Therefore, the number of flows affected in a loss event with j lost packets is equal to $max(n*(1-(1-(1/n))^j), 1)$.

Our studies have shown that this change is not necessary and can even be detrimental for a large value of n, and that it is a reasonable choice to enable it only for $1 < n < 13$ (which was done for all the presented results unless otherwise noted). This range, and the improvement attained therein, is also shown in Fig. 7.

A. Validation

We will start the validation by observing MulTFRC in isolation. To examine whether MulTFRC shows the same results as the equation, we ran the same simulations used for the equation validation, but instead of n parallel TCP flows, a single MulTFRC flow with the parameter n was used. The results for a random loss model on the shared link are shown in Fig. 4(a). The improved j calculation was introduced so that MulTFRC can better cope with more correlated loss, assuming that more than one lost packet can belong to the same virtual flow. On the other hand, these simulations had a noncorrelated loss model on the middle link, and this causes loss to be equally distributed among flows. As expected, MulTFRC with the improved j calculation obtains a smaller value of j, and therefore it sometimes achieves a significantly (even up to 15%) higher rate than TCP. Note that these were isolated tests of TCP and MulTFRC. Because the goal is not to always send as much as TCP, but to be as TCP-friendly as n TCPs would be, this behavior is not a problem, and perhaps even desirable, *in the absence of TCP*. As the loss rate grows (and conformant behavior becomes more important), the probability that more than one lost packet belongs to the same flow grows and the difference between TCP and MulTFRC becomes smaller. For loss rates between 5% and 8%, the difference is in the range between 0.1% and 5%.

The results with a DropTail queue on the shared link and without any artificial loss are shown in Fig. 4(b). MulTFRC and TCP obtained almost exactly the same rate. The RED queue has a different influence on the MulTFRC flow than on the TCP flows; this is visible in Fig. 4(c). A MulTFRC sender sends packets spread over an RTT and TCP sends packets in bursts.

For bottleneck capacities below 128 Mb/s, the rate of MulTFRC is between 0.7% and 7% lower than TCP irrespective of the parameter n. For the bottleneck capacities 128 and 256 Mb/s, the throughput of MulTFRC is a bit higher than the throughput of the TCP flows if n is small (about 2% for the $n = 5$ case shown in the figure). On the other hand, a large n and a bottleneck capacity of 128 or 256 Mb/s cause MulTFRC to gain a lower rate than TCP [Fig. 4(c)]. This is related to our choice of the queue size. MulTFRC, which is sending with a rate close to the bottleneck bandwidth, experiences loss in a number of consecutive RTTs and reduces its sending rate drastically. On the other hand, TCP backs off after the first loss is detected. We also investigated the same scenario, but using the queue size used for TFRC validation in [2]. Here, the results are quite different: In this case, MulTFRC obtains a slightly higher rate even for large values of n, and it is able to saturate the middle link.

Fig. 6 illustrates that because having a single "n-TCP-friendly" flow eliminates the potential of individual TFRC flows to harm each other, MulTFRC reacts faster to congestion than TFRC. The figure shows the throughput over time for a flow traversing a 15-Mb/s, 20-ms RED bottleneck link with periodic loss from an ns-2 simulation. At the beginning, we set loss to 1%, at 5 s we increased it to 10%, and at 14 s we changed it to 0.5%. To compare our protocol to TFRC, we ran five separate simulations: one TFRC flow, two TFRC

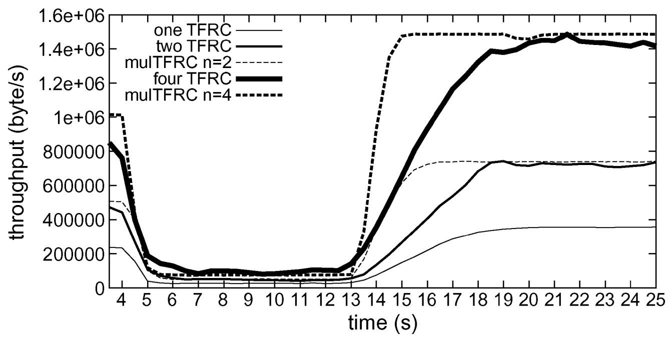


Fig. 6. Dynamic behavior of MulTFRC.

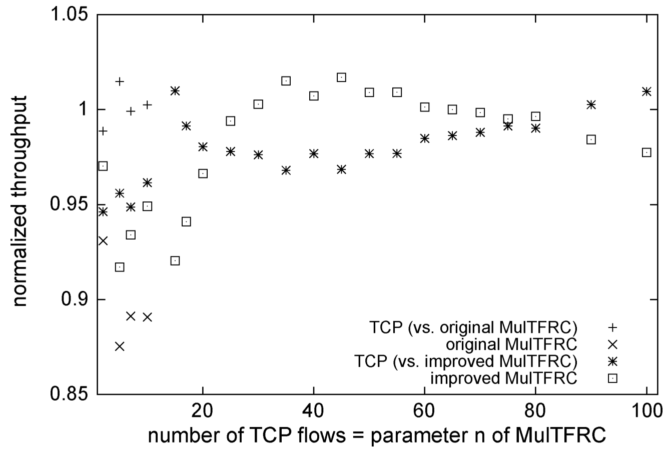


Fig. 7. TCP-friendliness of MulTFRC, with old and new calculation of j for $n < 13$.

flows at the same time, four TFRC flows at the same time, one MulTFRC flow with $n = 2$, and one MulTFRC flow with $n = 4$. It can be seen that the responsiveness of MulTFRC does not change as n increases.

Now, we will evaluate the "n-TCP-friendliness" of MulTFRC. The network configurations used in the following simulations are based on the simulations used in [2], i.e., the same network parameters were used. The topology is a dumbbell. All queues, except for the queue of the bottleneck link, implement the DropTail queuing mechanism. Access links have a 10-Gb/s capacity and 2-ms delay on the sender side of the bottleneck link, and 1-ms on the receiver side of the bottleneck link. The throughput of MulTFRC and TCP are presented as normalized values, where the average throughput of TCP flows and the throughput of MulTFRC were divided by the fair share of the bandwidth of individual flows—bandwidth divided by $2n$. The throughput of MulTFRC is also divided by n. Ideally, these values should be 1, meaning that flows obtain their fair share of the bandwidth.

We evaluated MulTFRC with a range of values for n. In each simulation, a MulTFRC flow shared the bottleneck link with n TCP flows. Using a 64-Mb/s/20-ms RED bottleneck link, Fig. 7 shows that the possible range of values for n can be very large without sacrificing n-TCP-friendliness. All values are between 0.89 and 1.01.

As a next step, the characteristics of the bottleneck link were varied, but care was taken that this link was always the real bottleneck of the network. For the first set of simulations, the bottleneck link implements the RED queue mechanism. The bottleneck link delay was 20 ms, and the capacity was changed from 10 to 70 Mb/s. The RED queue parameters were the same as in [2]. The impact of MulTFRC on TCP with a RED queue is shown in Fig. 8(a). The utilization of the bottleneck link was above 93%, and TCP always had more than 90% of its fair share—in most cases, around 99%.

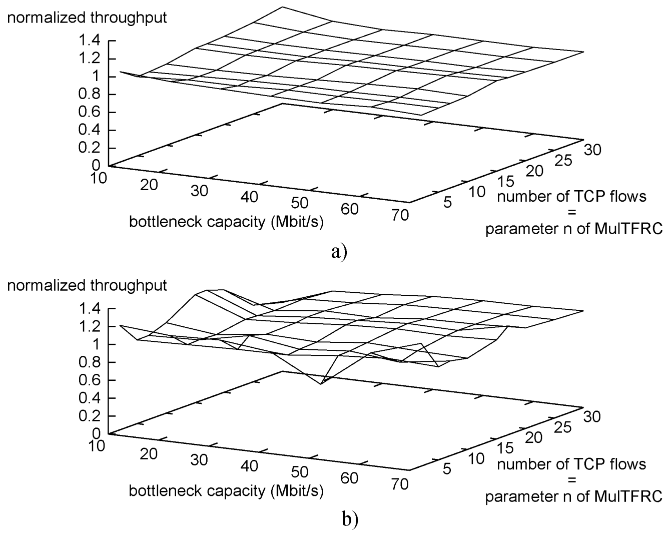


Fig. 8. TCP under the influence of MulTFRC. (a) RED queue at the bottleneck. (b) DropTail queue at the bottleneck.

A relatively low TCP fair share of the link is not always a sign of MulTFRC being more aggressive. In most such cases, there was a lower overall link utilization. The difference between the throughput of TCP and MulTFRC was almost always less than 5% of the throughput of the TCP flows. It only exceeded 10% in some simulations with a 10-Mb/s bottleneck link. For this bottleneck capacity, whether TCP or MulTFRC achieved a higher rate depends on n. In the range where MulTFRC had a higher rate, TCP had around 90% of its fair share, which means that MulTFRC does not drastically influence the TCP flows. With $n = 30$, high loss was seen, and TCP had a higher rate than MulTFRC; TCP flows obtained 106% of their fair share and MulTFRC had only around 82%. For this simulation, the loss rate was above 14%.

We ran similar simulations with a DropTail queue on the bottleneck link. Its delay was set to 20 ms, and the capacity was varied from 10 to 70 Mb/s (the same as in the previous simulation set). As in [2], the buffer of the DropTail queue was set to three $\times$ bandwidth $\times$ delay. To avoid phase effects, we changed the delay with values between 0 and 2 ms on the receiver side. As Fig. 8(b) shows, MulTFRC does not significantly affect the throughput of TCP flows and, in cases of notable mismatch, TCP almost always obtained a larger rate (MulTFRC only had a significantly larger rate for a capacity of 50 Mb/s and $n = 2$). The high link utilization was seen here as well (always above 97%, mostly larger than 99%).

MulTFRC almost always had more than 90% of its fair share of the bandwidth. Only for the bottleneck capacity of 10 Mb/s and n equal to 2, 15, 17, and 20, MulTFRC showed poor behavior, and it obtained just around 75% of its fair share. This can be due to a phase effect that could not be completely eliminated. For the other capacity values, MulTFRC obtains a more

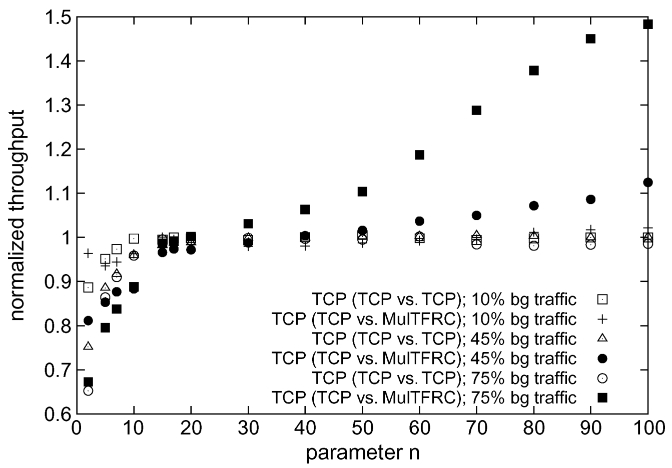


Fig. 9. TCP-friendliness of MulTFRC with Web-like background traffic.

than 7% lower rate than the TCP flows only for very low loss rates, which is due to the error introduced by the model discussed in Section II-C.

We also studied MulTFRC in the presence of Web-like background traffic. We investigated how MulTFRC influences TCP with such a background load by comparing the throughput of TCP flows when they are sharing the link with MulTFRC and when, instead of MulTFRC, a corresponding number of other TCP flows are present. We ran two sets of simulations. The first set had the same setup as in the previously shown simulation, but Web traffic was introduced. In the second set of simulations, n TCP flows shared the link with n other TCP flows instead of MulTFRC (i.e., there are $2n$ flows, but only the first n TCP flows are of interest; we will call them "observed flows"). In both sets of simulations, the background traffic occupied 10%, 45%, and 75% of the bandwidth in three separate runs. We used the PackMime-HTTP traffic generator for ns-2 [8].

The same normalization was used as in Fig. 7, but here the available bottleneck bandwidth was decreased by the bandwidth that is occupied by the background traffic. Therefore, also here, the ideal normalized throughput is 1. In Fig. 9, we show the normalized throughput of the observed TCP flows in the simulations with MulTFRC and the observed TCP flows competing with TCP. MulTFRC always occupied the remaining bandwidth well, and therefore almost always achieved a normalized throughput close to 1. In general, there was no notable difference between the rate achieved by TCP flows competing with MulTFRC and TCP flows competing with TCP flows. Only for a high loss rate, TCP obtained a significantly higher throughput when it competed with MulTFRC. This is due to the low rate of MulTFRC in the presence of very high loads of Web-like traffic, which we believe to be caused by its slowly responsive behavior.

To evaluate MulTFRC with $n \in \mathbb{R}^+$, we ran a MulTFRC flow with n changing from 0.1 to 2.0 against a single TCP flow. In all simulations, a RED queue was used at the 8-Mb/s bottleneck link, which had a delay of 20 ms; the same RED parameters were used as before. The simulations lasted 200 s, and the last 185 s were used for throughput measurements. The throughput was calculated by looking at the number of packets arriving at the receiver, irrespective of their sequence number. Each simulation setup was run 10 times, and the average of these runs was normalized against the throughput of the TCP flow such that the throughput of TCP is always equal to one.

As a result (shown with a diagram in [3]), the difference between the normalized throughput of MulTFRC and its ideal value of n (0.1, 0.2, etc.) was never more than 4% of n, except for $n = 0.1$ and $n = 0.2$. Here, the difference was around 15%. By observing each simulation separately, we found that the difference was greater than 10% of n in more than two runs only for $n \leq 0.3$, and in further 10 individual cases for n between 0.4 and 1.9 (in some cases, this was because the loss rate was very small, and in such cases, MulTFRC sometimes obtains a lower fair share of the bottleneck capacity). The standard deviation of the difference between MulTFRC and n was smaller than 4% of n. This value is only larger for $n = 0.1$ to 0.3.

B. Evaluating MulTFRC Against Prior Approaches

Just like MulTFRC, the Coordination Protocol (CP) was developed on the basis of TFRC, with the intention of behaving like n TFRC flows. The underlying idea is to multiply the equation from [1] with n in the protocol. As the authors of [4] have shown, and as we have discussed at the beginning of this paper, this is not enough because the loss-event rate of a single flow share is not the same as the loss-event rate of the cumulative flow. They therefore extended this concept by considering the n emulated flows as so-called "flow shares" and using a stochastic technique for filtering packets. The goal of this method is to determine which packets belong to a single flow share, and then use this flow share to calculate the loss-event rate. For CP, it is explicitly assumed that n is always greater than 1. Therefore, it would not work for $0 < n < 1$.

Because of using a stochastic technique, this protocol fluctuates more than ours. This is shown in Fig. 10(a), which was generated using the original code that was provided by the authors of CP. MulTFRC with $n = 4$ and CP with four flow shares traversed the a 15-Mb//s, 20-ms RED bottleneck link with periodic loss of 2% (MulTFRC and CP were run in separate simulations). The figure also shows the cumulative rate of four TCP and TFRC flows for comparison. Clearly, the rate of MulTFRC fluctuates less than the others (and CP is worse than four TFRCs).

Since CP is based on multiplying the equation in TFRC with n, any error that the equation produces will be amplified as n grows. On the other hand, because of the nature of our equation from Section II, our protocol works even better with an increasing number of flows. Fig. 10(b) shows results of running CP against a number of TCPs and running MulTFRC against a number of TCPs with the simulation setup that we already used for Fig. 7, but with a bottleneck capacity of 32 Mb/s (this figure shows the throughput of TCP divided by the throughput of CP or MulTFRC). With CP, multiple runs of the same setup yield significantly different results.

We tested MulTCP [9] in the same simulation setup as CP using an update that is available under "contributed code" via the main ns-2 Web page to make it work with the most recent version of ns-2. As stated in [9] and as our comparison shows, MulTCP performs reasonably well with values up to $n = 10$. This is shown in Fig. 10(c). The fact that MulTCP is a reliable protocol and lost packets are retransmitted does not have an influence on the results shown here, as we only study throughput at the transport layer (i.e., the number of packets arriving at the

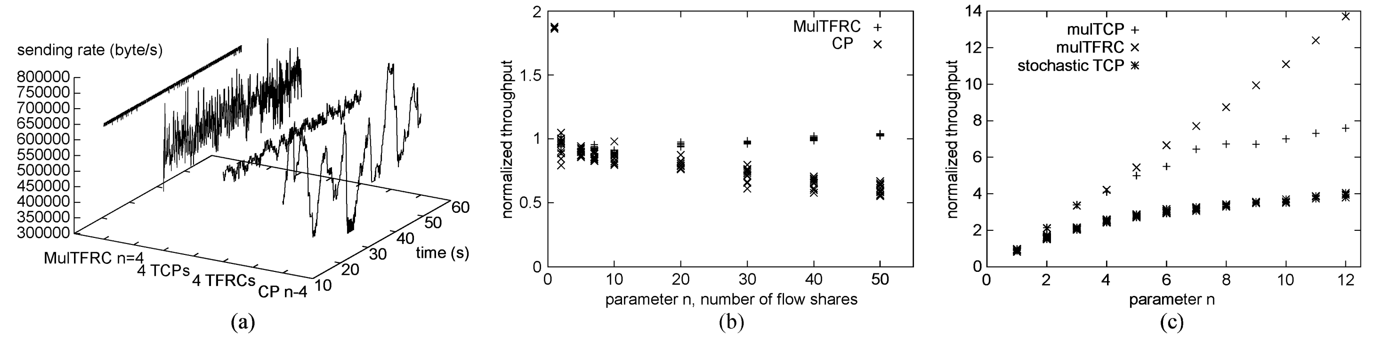


Fig. 10. Comparison to related work. (a) Rate fluctuations of MulTFRC, TCP, TFRC, and CP. (b) MulTFRC versus TCP and CP versus TCP. (c) MulTFRC, MulTCP, and Stochastic TCP.

receiver irrespective of their sequence number). This figure was generated using the same simulation setup as for Fig. 10(b), but without competing TCP flows. MulTCP shows some deficiencies. First, while it is obviously the most important feature of this protocol to emulate n TCPs as precisely as possible irrespective of the value of n, the possible choices of n that yield a satisfactory behavior are quite limited, and choosing $0 < n < 1$ is not possible. Second, MulTCP is not suitable for certain multimedia applications because of its severe rate fluctuations.

The figure also shows Stochastic TCP [10], which is another protocol that we tested using the code provided by the authors. This protocol, which partitions a single congestion window into a set of "virtual streams" that are stochastically managed as individual TCP streams, was built for high-speed networks. In our simulations with a smaller bandwidth, its performance was poor [as seen in Fig. 10(c), simulations with Stochastic TCP were run several times].

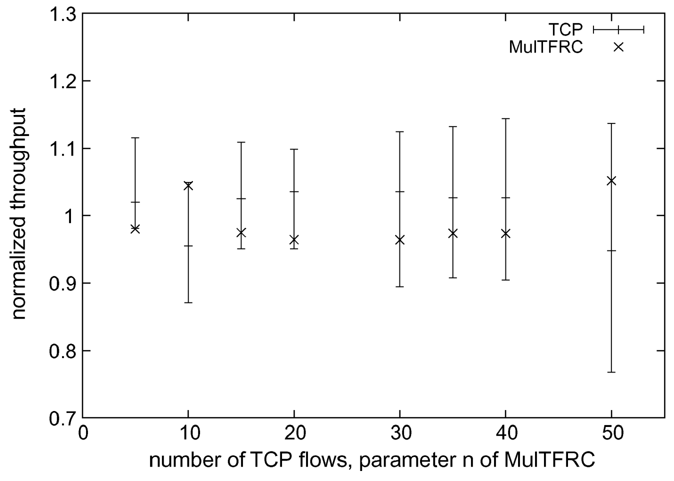


Fig. 11. MulTFRC versus TCP in local test bed.

C. Real-Life Evaluation

In addition to the ns-2 code of the MulTFRC protocol, we developed a real-life implementation. It is an extension of the original code provided by the TFRC authors,[5] which uses UDP as a transport-layer protocol. Rate control according to MulTFRC is thus implemented at the application layer. A minor mismatch between the simulation results and real-life measurements can therefore be produced by additional delay on the hosts.

The first tests were run in a local test bed. It consists of three hosts, one of which is acting as a router and the other two as sender and receiver for TCP flows and MulTFRC. All hosts run the Linux kernel ver. 2.6.17.1. We used the *tc* command to introduce a delay of 20 ms in both directions and to set the bandwidth to 32 Mb/s because MulTFRC is implemented in the application layer, and this fact can limit the maximal sending rate. A MulTFRC flow with parameter n and n TCP Sack flows were started at the same time. The packet size was 1448 for TCP and 1460 for MulTFRC (in the size, the header is not included). The throughput is measured as the number of packets sent per second.

For each measurement, the throughput of the TCP flows and the MulTFRC flow was normalized; it was divided by the average throughput of all observed flows (here, we assume that a MulTFRC flow consists of a number of individual flows). The throughput of the MulTFRC flow was also divided by n. Fig. 11 shows that the real-life implementation yields quite similar results as the ns-2 simulations shown in Fig. 7. The difference between the normalized throughput of TCP and MulTFRC is always less than 0.1, and it is largest for n equal to 10 and 50. The same can be seen in the simulations, but the difference is slightly smaller there. In Fig. 11, we also show the normalized minimal and maximal throughput obtained by individual TCP flows.

To study the responsiveness of the MulTFRC real-life implementation, we examined its behavior when TCP traffic joins or leaves the network. We ran two tests. In all tests, MulTFRC had n set to 4, and there were four TCP flows present in the network. The bottleneck link had a delay of 20 ms and a capacity of 15 Mb/s (the *tc* command was used). In the first scenario, the TCP flows and a MulTFRC flow started at the same time. After they reached stable rates, the TCP flows terminated. This is shown in Fig. 12(a). As it can be seen, the MulTFRC flow reacted fast and quickly utilized this new available bandwidth. Fig. 12(b) shows the opposite scenario. In the first 50 s, MulTFRC was the only flow in the network, and then four TCP flows started. Again, MulTFRC reacted fast.

To test MulTFRC on the Internet, we used PlanetLab. Since our implementation of MulTFRC uses UDP as the transport-layer protocol and we found that often, in case of congestion, UDP packets are dropped more frequently than TCP packets, a fair comparison was not always possible. The experiments were done on October 15, 2009 and between January 13–15, 2010.

[5]http://www.icir.org/tfrc/code.

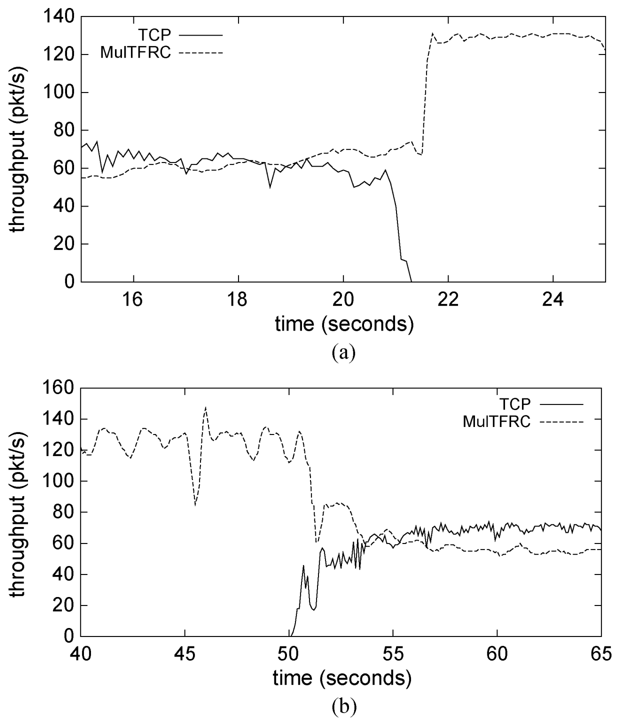


Fig. 12. Local test bed: MulTFRC responsiveness on TCP traffic. (a) TCP ends at the 21st second. (b) TCP starts at the 50th second.

MulTFRC with parameter n started at the same time as n TCP flows; the average throughput over 200 s was measured (omitting the first 60 s). The TCP flows had window scaling turned on, and the advertised window scaling value was 7. The same normalization as with the local test-bed measurements was used.

MulTFRC was run between the host in Innsbruck (the same host was used as in the validation of our models—138.232.65.6) and the already mentioned host in France (planetlab2.utt.fr). The corresponding results for n equal to 10, 15, and 20 are shown in Fig. 13(a). For $n = 10$, the rate of MulTFRC was between 3.2% and 9.7% larger than the rate of TCP. Also in the simulations Fig. 7, MulTFRC had a larger rate for $n = 10$. The rate of MulTFRC with $n = 15$ was between 0.8% and 7.8% lower than the rate of TCP. The results for $n = 20$ are very satisfactory too. Some tests showed a difference between TCP and MulTFRC of only 0.39% and 0.66% of the TCP throughput. The difference was never more than 6% of the TCP flows' throughput.

The second set of measurements was run between our host in Innsbruck and the previously used host in Portugal (planetlab1.fct.ualg.pt). Fig. 13(b) shows results of these measurements. Here, for some measurements, a greater difference was seen. For $n = 10$, there is a range from a small difference of only 4% of the TCP throughput to up to 16%. In most cases, MulTFRC had a slightly larger rate than TCP. The opposite is seen in only one case where MulTFRC had an 8% smaller rate. Similar results are seen for n equal to 15 and 20. For $n = 15$, the rate of MulTFRC was between 0.9% and 15% larger than the rate of TCP flows. For $n = 20$, only four measurements were performed because the host became unavailable. In two measurements with $n = 20$, MulTFRC had a large rate (2% and 11% more than TCP), and the other two times TCP flows had a 1% and 22% larger rate than MulTFRC.

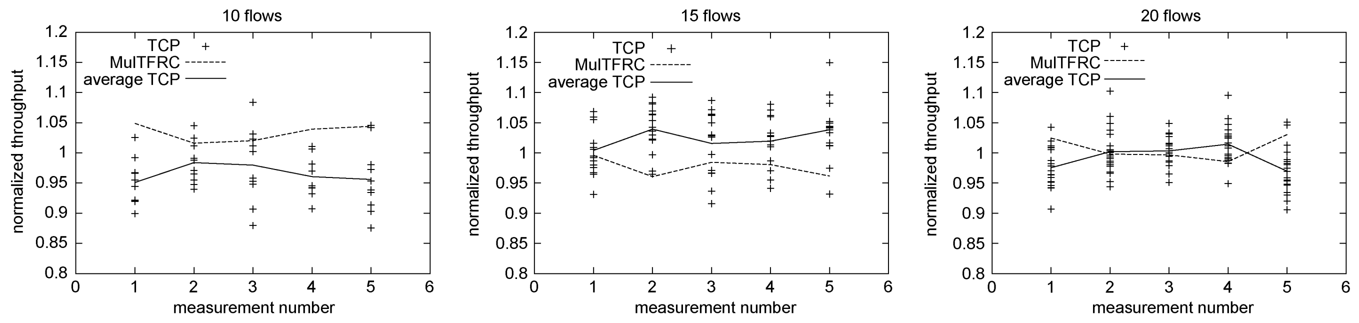
IV. Related Work

As we have initially stated, the model that we have based our work upon is rather old. There is now a wealth of TCP models available (e.g., [11]–[14]), many of which are better than the model in [1] in some aspect, but they are all much more complex, making their practical application much harder. Some effort was also made to model a number of TCP flows sharing a bottleneck instead of focusing on just a single one [14]–[21]. The usability of these works is limited because they assume that the flows under consideration are alone in the network. This is mainly due to loss being expressed as a function of the previous rate of these flows. To the best of our knowledge, only one model [22] precisely captures the behavior of a number of TCP flows in the presence of (loss from) other traffic. Being a dynamic model based on differential equations, it is hard to use in practice.

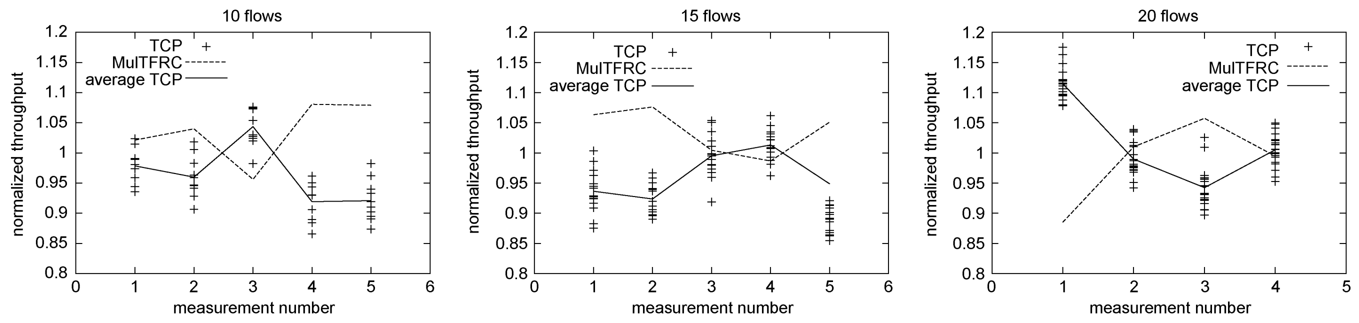
In what follows, we concentrate on work that is related to MulTFRC. We already discussed a few protocols that are related to ours: MulTCP, Stochastic TCP, and CP. Like CP, Probe-Aided MulTCP (PA-MulTCP) [23] was proposed as a means to control aggregates of end-to-end flows between two intermediate network nodes. As its name suggests, it is based on MulTCP and improves upon it by using probes. The goal of these probes is to determine a loss rate that is closer to the loss rate experienced by a real TCP than the loss rate of the original MulTCP. The loss rate that is normally experienced by MulTCP is smaller than the loss rate of n standard TCP flows, making MulTCP too aggressive. The probe-based loss rate measurement effectively adds another control loop to the protocol, which is used to impose an upper limit on the window size of the already existing (MulTCP-like) one. We did not run simulations with PA-MulTCP because the original code was not available and because its applicability is limited by the overhead of using an additional control loop. MulTFRC, on the other hand, was designed to be a general-purpose protocol with a broad range of usage scenarios.

MPAT [24] is another mechanism that is concerned with controlling aggregates. Unlike PA-MulTCP and CP, however, MPAT maintains the control loops of individual TCP flows and lets them share their congestion state. In the example in [24], if two TCP flows would be allowed to send five packets each, but the bandwidth should be apportioned according to a 4:1 ratio, MPAT could allow one flow to send eight packets and let the other flow send two. Such differentiation between flows is a way to provide quality of service (QoS) via congestion control, and this is also done in OverQoS [25], albeit with MulTCP. While we believe that MulTFRC could also be used for this purpose, we consider a discussion of such usage to be beyond the scope of this paper.

On the extreme end of "friendliness" toward the network, several mechanisms for so-called "less-than best effort" behavior (which is the same as our $0 < n < 1$ case, but with a possibly unknown value of n under this constraint) have been proposed. TCP Nice [26], for example, is a variant of TCP Vegas [27] with a more conservative congestion control behavior. Just like these two algorithms, TCP Low Priority (TCP-LP) [28] relies on increasing delay as an indicator of imminent congestion and defines a rate control method that lets the end system back off earlier than standard TCP.



(a)



(b)

Fig. 13. MulTFRC versus TCP on PlanetLab. (a) Innsbruck—France. (b) Innsbruck—Portugal.

Since increasing delay is an effect that can be measured earlier than packet loss or ECN marking, relying on it is a common theme in such work. This is not the case for MulTFRC with $0 < n < 1$. Therefore, our protocol is likely to be more aggressive than these alternatives. On the positive side, this aggression is tunable in MulTFRC.

We note that the name MulTFRC was used before in [29] and its follow-up work [30], [31] to describe a system that employs a number of separate TFRC connections to improve the performance of TFRC over wireless links. It was used in [32]–[34] to assess the impact on buffer management and video quality in WiMAX networks. We decided to keep our name because of its illustrative nature and its similarity: It should be extremely easy to replace the multiple real TFRC connection in this work with our scheme, thereby inheriting all its benefits and making it easier to use (with multiple connections, data must be striped into separate buffers, whereas only one connection is needed with our MulTFRC).

The approach of [29] has been criticized for obvious problems that stem from the use of multiple connections, e.g., in [35] ("…large rate fluctuation and slow convergence caused by the frequent changes in the number of simultaneous connections"). The authors accordingly improved their scheme in [36] by using only one connection with CP. As we have shown, our mechanism outperforms CP; it is also much easier to implement. Also based on [29], a loss discounting scheme is applied in [37] to achieve a finer-grained rate control. This appears to be a largely simplified and less performant variant of what is also done in CP [4]. One more effort in this direction is [38], where the sending rate is multiplied by the number of emulated flows. As we have already explained, this does not lead to the desired result.

To summarize, there appears to be a major interest in using a MulTFRC-like scheme for streaming media over wireless links. However, all the related work relies on a less performant mechanism that opens multiple separate TFRC flows or derivatives thereof. Our method is simpler to implement (essentially, all that is needed to turn TFRC into MulTFRC is to replace the equation and include a measure of the packet loss ratio in the feedback to the sender), easier to use, and performs better.

V. Conclusion

We have derived an extension of the well-known TCP model in [1] and validated it with simulations and real-life tests, showing that it works very well in a wide range of conditions despite its simplicity. We stated that, given the more precise yet more complex related work available, the value of our model is its practical applicability, which we proceeded to show by applying it ourselves, to change TFRC into MulTFRC.

Like TFRC, MulTFRC is a rate-based congestion control mechanism that is suitable for multimedia flows because of its smooth sending rate, but instead of being TCP-friendly, it is n-TCP-friendly. As we have shown with simulations and real-life experiments, both in a local test bed and across the Internet, MulTFRC supports a wide range of possible values for n, with $n \in \mathbb{R}^+$. This renders it fit for various purposes, e.g., also for unobtrusive "background" traffic in case of $0 < n < 1$.

We believe that MulTFRC should be experimented with on a wider basis to evaluate its performance under various Internet conditions, and we have made our code available for that purpose.[6] As for the equation, its applicability is certainly broad, and we hope that other interesting usage ideas will be found.

Acknowledgment

The authors would like to thank the colleagues who helped with [3], as well as S. Gjessing, W. Heiss, and M. Telek for valuable suggestions that contributed to this paper.

[6]http://heim.ifi.uio.no/michawe/research/projects/multfrc/index.html.

REFERENCES

[1] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, 1998, pp. 303–314.

[2] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM*, 2000, pp. 43–56.

[3] D. Damjanovic and M. Welzl, "MulTFRC: Providing weighted fairness for multimedia applications (and others too!)," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 3, pp. 5–12, 2009.

[4] D. E. Ott, T. Sparks, and K. Mayer-Patel, "Aggregate congestion control for distributed multimedia applications," in *Proc. IEEE INFOCOM*, 2004, vol. 1, pp. 13–23.

[5] E. Brosh, G. Lubetzky-Sharon, and Y. Shavitt, "Spatial-temporal analysis of passive TCP measurements," in *Proc. IEEE INFOCOM*, 2005, vol. 2, pp. 949–959.

[6] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[7] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," RFC 5348, Sep. 2008.

[8] J. Cao, W. S. Cleveland, Y. Gao, K. Jeffay, F. D. Smith, and M. C. Weigle, "Stochastic models for generating synthetic HTTP source traffic," in *Proc. IEEE INFOCOM*, 2004, vol. 3, pp. 1546–1557.

[9] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing TCP," *SIGCOMM Comput. Commun. Rev.*, no. 3, pp. 53–69, 1998.

[10] T. J. Hacker and P. Smith, "Stochastic TCP: A statistical approach to congestion avoidance," in *Proc. PFLDnet*, Manchester, U.K., Mar. 2008.

[11] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 356–369, Apr. 2005.

[12] V. Misra, W. Gong, and D. F. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to red," in *Proc. ACM SIGCOMM*, 2000, pp. 151–160.

[13] F. Baccelli and D. Hong, "Interaction of TCP flows as billiards," *IEEE/ACM Trans. Netw.*, vol. 13, no. 4, pp. 841–853, Aug. 2005.

[14] Y. Liu, F. Lo Presti, V. Misra, D. Towsley, and Y. Gu, "Fluid models and solutions for large-scale IP networks," in *Proc. ACM SIGMETRICS*, 2003, pp. 91–101.

[15] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel TCP sockets: Simple model, throughput and validation," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.

[16] F. Baccelli, D. R. McDonald, and J. Reynier, "A mean-field model for multiple TCP connections through a buffer implementing RED," *Perform. Eval.*, vol. 49, no. 1–4, pp. 77–97, 2002.

[17] C. Casetti and M. Meo, "A new approach to model the stationary behavior of TCP connections," in *Proc. IEEE INFOCOM*, 2000, vol. 1, pp. 367–375.

[18] L. Muscariello, M. Mellia, M. Meo, M. A. Marsan, and R. L. Cigno, "Markov models of Internet traffic and a new hierarchical MMPP model," *Comput. Commun.*, vol. 28, no. 16, pp. 1835–1851, 2005.

[19] M. Garetto, R. Lo Cigno, M. Meo, and A. M. Marsan, "Modeling short-lived TCP connections with open multiclass queuing networks," *Comput. Netw.*, vol. 44, no. 2, pp. 153–176, Feb. 2004.

[20] M. A. Marsan, M. Garetto, P. Giaccone, E. Leonardi, E. Schiattarella, and A. Tarello, "Using partial differential equations to model TCP mice and elephants in large IP networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 6, pp. 1289–1301, Dec. 2005.

[21] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan, "A detailed and accurate closed queueing network model of many interacting TCP flows," in *Proc. IEEE INFOCOM*, 2001, vol. 3, pp. 1706–1715.

[22] M. A. Marsan, M. Garetto, P. Giaccone, E. Leonardi, E. Schiattarella, and A. Tarello, "Using partial differential equations to model TCP mice and elephants in large IP networks," in *Proc. IEEE INFOCOM*, 2004, vol. 4, pp. 2821–2832.

[23] F. Kuo and X. Fu, "Probe-Aided MulTCP: An aggregate congestion control mechanism," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 1, pp. 17–28, 2008.

[24] M. Singh, P. Pradhan, and P. Francis, "MPAT: Aggregate TCP congestion management as a building block for internet QoS," in *Proc. ICNP*, Berlin, Germany, 2004, pp. 129–138.

[25] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "OverQoS: An overlay based architecture for enhancing Internet QoS," in *Proc. NSDI*, San Francisco, CA, 2004, p. 6.

[26] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 329–343, 2002.

[27] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, 1994, pp. 24–35.

[28] A. Kuzmanovic and E. W. Knightly, "TCP-LP: Low-priority service via end-point congestion control," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 739–752, Aug. 2006.

[29] M. Chen and A. Zakhor, "Rate control for streaming video over wireless," in *Proc. IEEE INFOCOM*, 2004, vol. 2, pp. 1181–1190.

[30] M. Chen and A. Zakhor, "Flow control over wireless network and application layer implementation," in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.

[31] M. Chen and A. Zakhor, "Multiple TFRC connections based rate control for wireless networks," *IEEE Trans. Multimedia*, vol. 8, no. 5, pp. 1045–1062, Oct. 2006.

[32] S. Saleh and M. Fleury, "Multiple TFRC streaming in a WiMAX environment," in *Proc. 7th IEEE CCNC*, Piscataway, NJ, 2010, pp. 953–957.

[33] S. Al-Majeed and M. Fleury, "Multi-connection TFRC video streaming in a WiMAX environment," in *Proc. MCIT*, 2010, pp. 117–120.

[34] S. Saleh and M. Fleury, "Video streaming with multi-TFRC and uplink queue management," in *Dig. Tech. Papers ICCE*, 2010, pp. 75–76.

[35] G. Yang, L.-J. Chen, T. Sun, M. Gerla, and M. Sanadidi, "Real-time streaming over wireless links: A comparative study," in *Proc. 10th IEEE ISCC*, 2005, pp. 249–254.

[36] M. Chen and A. Zakhor, "AIO-TFRC: A light-weight rate control scheme for streaming over wireless," in *Proc. Int. Conf. Wireless Netw., Commun. Mobile Comput.*, 2005, vol. 2, pp. 1124–1129.

[37] X. Tong and Q. Huang, "MULTFRC-LERD: An improved rate control scheme for video streaming over wireless," in *Proc. Adv. Multimedia Inf. Process.—PCM*, Oct. 2004, pp. 282–289.

[38] P. Yogesh, S. Bose, and A. Kannan, "Effective TCP friendly rate control scheme for video streaming over wireless networks," *Asian J. Inf. Technol.*, vol. 5, no. 4, pp. 442–447, 2006.

Dragana Damjanovic received the Ph.D. degree in computer science from the University of Innsbruck, Innsbruck, Austria, in 2010.

Since 2006, she has been a Research Assistant with the University of Innsbruck in a research team on Network Support for Grid Computing, which is a part of the Distributed and Parallel Systems Group. Before joining the University of Innsbruck, she spent two years as a Research Assistant with the Institute for Information Systems, University for Health Sciences, Medical Informatics and Technology (UMIT), Hall in Tirol, Austria.

Michael Welzl received the Ph.D. degree with distinction from the Technical University of Darmstadt, Darmstadt, Germany, in 2002, and received his habilitation from the same university in 2007.

He spent two years as a Research Assistant with the University of Linz, Linz, Austria, before joining the Institute of Computer Science, University of Innsbruck, Innsbruck, Austria, in November 2001, where he led a research team on network support for grid computing. Since May 2009, he has been an Associate Professor with the Department of Informatics, University of Oslo, Oslo, Norway.