Dragana Damjanović

Parallel TCP Data Transfers: A Practical Model and its Application

submitted to

the Faculty of Mathematics, Computer Science and Physics, the University of Innsbruck
in partial fulfillment of the requirements for the degree of doctor of science

advisor:

Prof. Dr. Michael Welzl

Innsbruck, February 2010

Abstract

The TCP protocol is the most used transport protocol in the Internet today. Since its first definition in 1981 and extension in 1988, when congestion control was added, the essence of its behavior has not changed. During the last years, network technology has been developing fast and TCP has begun to experience its limitations. One single standard TCP flow is not able to fully utilize modern high-speed links, therefore the number of applications using parallel flows grows.

Examining and modeling the behavior of single or multiple TCP flows has been the subject of a great amount of work in the past years, starting from very simple models to very complex ones. This thesis introduces models which strike a balance between precision and ease of use. The goal was to develop equations which estimate the throughput of parallel TCP flows that share an end-to-end path. The throughput calculation depends on the network conditions, which are primarily characterized by packet loss. Since this metric can be measured in different ways, by considering the packet loss rate of the whole traffic or the individual flows that it consists of, there is one equation for each of these two cases. Validation results involving simulations as well as real-life measurements show that both equations are highly accurate.

The fact that these models are easy to use makes them widely applicable. One of them is applied in the MulTFRC protocol, which is another major contribution of this thesis. As performance evaluations show, MulTFRC very accurately extends the well-known TFRC protocol from emulating only one to several TCP flows. Other possible applications of the models are also discussed.

Table of Contents

List of Figures

Chapter 1

Introduction

Nowadays the TCP protocol is the most used transport layer protocol in the Internet. The first specification of this protocol dates back to 1981 [105], but the most interesting aspect that dictates its behavior — congestion control — was added in 1988 [67]. Since then TCP has only undergone minor changes, but at the same time, link as well as application layer technology have changed drastically. Hence the TCP congestion control algorithm that over the years kept the Internet stable reveals its limits, e.g., by not being able to fully utilize high-speed links, not being suitable for wireless links etc. Being more aggressive than a single flow, parallel TCP flows simply achieve a higher link utilization and also a higher throughput. Therefore transfers over multiple connections came as a simple but rather not perfect step forward for achieving a better performance of applications (e.g. peer-to-peer application, GridFTP, bbcp etc). Parallel TCP flows are the central theme of this work.

Congestion control is a mechanism for protection against congestion collapse of a network by controlling the rate of the network. It is the most studied aspect of the TCP protocol since it determines the instant sending rate and thereby the most important measure of a connection's performance — throughput. Capturing dynamics of TCP congestion control has been the main subject of a great amount of work; a single TCP flow as well as multiple flows have been observed, and a broad range of assumptions about the underlying network and the loss distributions can be found in the literature. Nevertheless it seems that there are angles from which TCP has not yet been looked at, like the one presented in this thesis.

Here the modeling work concentrates on a number of parallel TCP flows sharing an end-to-end path. It should be noted that this model does not capture the behavior of parallel flows connecting the same end hosts but traversing different paths (e.g., when using traffic engineering with "Equal-Cost Multi-Path" — ECMP). The goal is to introduce a model which strikes a balance between precision and ease of use. Therefore details about the network topology and other coexisting traffic are neglected by just looking at the network from the source perspective.

The network, more precisely the end-to-end path, is characterized just by the packet loss and the round-trip time experienced by the flows. As an outcome, the modeling work provides two simple equations for calculating the throughput of parallel flows.

Parallel flows can be observed from two perspectives: examining each flow separately or all flows as an aggregate. Accordingly there are two different loss measures: a per-flow loss measure and a loss measure of the cumulative flow. Without having a need for separating the packets into flows and assuming all of them to form a single cumulative flow, measuring is easier. Nevertheless, since the model with a per-flow loss measure assumes a broader knowledge about loss experienced by flows and therefore could produce more precise results in case the necessary information is available, one equation is developed for each of these two loss measures. Validation results involving simulations using the ns-2 simulator [4] as well as real-life measurements show that both equations are highly accurate.

Owing to their simplicity, these models show an immediate practical benefit. The MulTFRC protocol, the second contribution of this thesis, is an example. Other possible applications of the models are also discussed.

MulTFRC is based on TCP-Friendly Rate Control (TFRC) [50, 51]. It is a rate-based protocol and, similar to the way TFRC uses the equation from [103], MulTFRC uses one of the throughput equations introduced above to calculate the flow's sending rate depending on the network conditions (the packet loss measure and the round-trip time). Since the equations derived here model the throughput of a number of flows instead of a single one, an additional parameter, number of flows — n, is available. The sending rate of a MulTFRC flow is comparable to the sending rate of a number of standard TCP flows. As simulations carried out using the ns-2 simulator [4] and a real-life implementation show, MulTFRC provides a desirable behavior for a rather wide range of possible values of n including not just natural numbers but also real numbers and values between one and zero.

"TCP-friendliness" [91, 93] describes a requirement for a protocol to guarantee that the steady-state rate of a flow using it is comparable to the one of a conforming standard TCP flow. MulTFRC behaving as n standard TCP flows is "n-TCP-friendly".

MulTFRC is suitable for multimedia traffic since it provides a smoother rate, which is of primal interest for such an application, but the service that it offers makes it attractive for other scenarios, too. Therefore a reliable implementation of MulTFRC is provided in the form of a simple tool for a file transfer. TCP's congestion control algorithm alone, in despite of used queuing mechanisms, provides a certain form of fairness between flows. The rate allocation is approximately proportionally fair [70, 121] and all TCP flows receive a fair share of links, although the user's benefit from each flow is not the same (e.g., the user could be more eager to obtain file A than file B). Since MulTFRC emulates n TCP flows, it provides weighted proportional fairness between flows where weights correspond to a chosen value for parameter n. In this way it gives users the possibility of assigning weights to their flows depending on personal benefit. Further examples of the flexibility it offers and its applicability are also provided.

Our modeling work presented in this thesis can also be found in the form of a technical report [39] and it was presented as a poster at SIGCOMM 2007 (the abstract describing the poster is [37]). MulTFRC is described in [38] and an Internet-Draft of the protocol is provided [125] (the Internet-Draft is also given in Appendix B). MulTFRC was also presented at the meeting of the IRTF "Internet Congestion Control Research Group" working group at the 75th IETF meeting in Stockholm, Sweden and at the meeting of the IETF "Datagram Congestion Control Protocol" working group an the 76th IETF meeting in Hiroshima, Japan.

1.1 Structure of the dissertation

There are two main contributions of this thesis, and a distinct separation between the two is to be seen throughout the structure.

After the short introduction provided here, the work naturally starts by addressing the main technical details necessary for the understanding of further discussions as well as by taking a closer look at the existing literature touching themes relevant for this work (Chapter 2). TCP being the central topic of this thesis, for completeness and better understanding of this work, a short description of its most relevant aspects is provided in Section 2.1. It is followed, in Section 2.3, by a discussion of the modeling work existing in the literature. Further, Section 2.5 deals with other proposed transport layer protocols with special attention given to the protocols offering a "n-TCP-friendly" or a "less-than-best effort" service. For a better understanding of the fairness problem a short overview of the existing discussion about this theme is outlined.

The rest of the thesis is divided into two parts: 1) modeling of parallel TCP flows (Chapter 3) and 2) the MulTFRC protocol design (Chapter 4). Then, Chapter 5 concludes this work.

As already mentioned, two models are developed, and consequently Chapter 3 consists of: a description of the model with the per-flow loss measure (Section 3.2) and the model with the loss measure of the cumulative flow (Section 3.3). Each model's derivation is followed by comprehensive validations using simulations as well as real measurements. The chapter ends with a short discussion of their applicability.

The MulTFRC protocol uses one of these equations. The design of this protocol is discussed in Chapter 4. The final algorithm is given in Appendix A. The protocol design is followed by its evaluation and comparison with relevant related work (Section 4.2). The real-life implementation of MulTFRC, presented in Section 4.3, produces similar results. Our point of view about the usefulness of the protocol composes Section 4.4. In addition, an Internet-Draft specifying this protocol is provided in Appendix B.

Chapter 2

Related Work

This thesis consists of two main parts. It starts with the modeling of TCP flows, described in Chapter 3. Using these models, MulTFRC is developed and its description is given in Chapter 4. For a better understanding of this work, we will introduce related work in both areas, in Sections 2.3 and 2.5 respectively.

In Chapter 3 of this work, we analyze one of the TCP aspects, i.e., the throughput of parallel TCP flows sharing the same path. Therefore the TCP modeling efforts presented throughout the literature certainly need our attention. Section 2.3 gives a broader overview of mathematical analysis of TCP beyond concentrating only on the models of multiple TCPs. Since the mathematical approaches and especially the assumptions taken in the case of single flow models are an interesting aspect, they are also covered in this section. A discussion about relationships between these models and our model is to be noticed throughout this section and it is summarized at the end of it.

The second outcome of this thesis is MulTFRC (Chapter 4). This protocol is based on TCP-Friendly Rate Control (TFRC) protocol and therefore a short introduction of this protocol is given in Section 2.4. MulTFRC is a protocol with weighted congestion control. Consequently, a closer look is taken at the existing approaches for achieving weighted congestion control (Section 2.5). As proved later in this thesis (Chapter 4), a chosen weight for MulTFRC's congestion control can be any positive real number, even less than one. In such a case, a flow would be less aggressive than a standard TCP flow, and therefore, an overview of protocols providing a "less-than-best effort" service is given at the end of Section 2.5. With weighted congestion control, weighted proportional fairness can be achieved; hence in Section 2.6 we try to present the idea behind weighted proportional fairness and also touch the current discussion about fairness in the Internet.

Since, as already mentioned, the TCP protocol is the main focus of this work, this chapter starts with a short introduction to TCP (Section 2.1). Queuing mechanisms are an important aspect of the network, and understanding their influence on traffic is crucial for interpreting results presented in this thesis. We will therefore also give an short overview of the relevant queueing mechanisms in Section 2.2.

2.1 The Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is a transport layer protocol that provides a reliable communication service on top of IP networks. The first specification of the TCP protocol dates back to 1981 [105], but the most interesting and broadly studied aspect of the TCP protocol was introduced in 1988 — congestion control [67]. In the following years TCP has only undergone some minor changes. Congestion control is a mechanism for protecting a network against "congestion collapse", which refers to a state of the network where any further increase of sending rates yields a decrease of the achieved throughput; the Internet experienced its first congestion collapse in 1986 [67]. The congestion control mechanism was developed with the goal of avoiding congestion in the network by controlling the rate of the flows in the network. Since the most important aspects of a transfer — the rate of a sender and thereby the duration of the transfer are regulated by the congestion control algorithm, an analysis of the behavior of TCP mainly becomes a study of the congestion control algorithm.

Data of a transfer using TCP are divided into units called packets; an alternative name is segments. The TCP protocol enables a reliable transfer of data between two end points, and to do so, first a logical connection between these two points is established (the so-called connection establishment phase). This is accomplished using a three-way handshake: assuming host 1 initiates a communication with host 2, host 1 sends a packet with the SYN flag set (SYN stands for synchronization) and host 2 responds with a SYN/ACK packet (ACK — acknowledgement). As the last step, host 1 replies with an ACK. Reliability is ensured by acknowledgements (ACK) — the receiver confirms the correct reception of each sent packet by sending an acknowledgment. More precisely, TCP uses cumulative acknowledgments, which means: by an ACK acknowledging packet k, all packets till and including the $(k-1)$th packet are confirmed. If a packet is lost or a corrupted packet is received, the receiver repeats the ACK for the previous packet, therefore a reception of duplicate ACKs at the sender is a sign of a packet loss and consequently the sender retransmits the missing packet. Loss of packets can also be suspected if ACKs are not received for a certain duration of time, assuming all packets or acknowledgements have been lost. These ways of detecting packet loss are also used for inferring the existence of congestion in the network.

In addition to the already mentioned congestion control, TCP also implements flow control. The aim of the flow control is to protect the receiver against overflow. TCP has a window based flow control, i.e., a TCP flow is allowed to send a certain amount of data, denoted by "window", before feedback acknowledging the correct reception of the data is received. The

receiver constrains the sender by granting a certain "window". The allowed window size is conveyed through each ACK as a value set in a field of the header. Because of the size of the header field this value is limited to 64KB and most of the connections in today's Internet would be limited by this value. Therefore the window scaling option was introduced. The option is sent with SYN and SYN/ACK packets and it indicates that an end host is willing to apply this feature and, at the same time, it specifies a host's desired scaling factor. This value is used for scaling the receiver window. Unless otherwise noted, the receiver window is assumed to be large enough such that it does not limit a transfer in this thesis.

The congestion control algorithm influences the sending rate of a TCP flow by prescribing the change of what is called the "congestion window" (cwnd). The rate of the TCP flow is bound by the minimum of the congestion window and the window allowed by the receiver. In the context of this thesis, "window" refers to cwnd unless otherwise noted. The TCP protocol implements an additive increase multiplicative decrease (AIMD) congestion control algorithm. The congestion window is additively increased in the absence of congestion and multiplicatively decreased upon detecting congestion. The congestion control algorithm slightly differs depending on the TCP variant which is considered. Since TCP Reno is used for the modeling work presented in this thesis, its congestion control algorithm will be described here.

The window growth is divided into two phases with different increase functions:

- **the slow start phase**
- **the congestion avoidance phase**

In the slow start phase the size of the congestion window is increased exponentially, which is achieved by increasing it by the size of one data unit for each received acknowledgement. This yields a doubling of the size of the congestion window every round-trip time (the round-trip time is the time needed for a packet to travel from the sender to the receiver and back). In the congestion avoidance phase the congestion window is increased by $MSS*MSS/cwnd$ (MSS — Maximum segment size, i.e., the size of the TCP data unit) for each received ACK, which approximately increases it by one data unit every round-trip time. Not all implementations follow exactly this rule; they conform to the standard as long as *cwnd* is not increased by more than one MSS per RTT.

TCP is informed about congestion by packet loss. A set of losses on a TCP window causes TCP to reduce its rate, and such an event is called "congestion event" or "loss event". Loss can be detected in two ways:

- **receiving three duplicate acknowledgements (DupACK)** — as already mentioned, receiving duplicate ACKs is a sign of packet loss. To be more robust to packet perturbation in the network, the TCP sender waits for three duplicate ACKs before deducing that the network is congested. Then, TCP enters the "fast retransmission/fast recovery phase" (explained below). In this thesis, detecting loss in this way will be referred to as "triple-duplicate loss events".

- **receiving no acknowledgments for a certain duration of time, i.e., retransmission time-out RTO** — RTO is set dynamically depending on the round-trip time and its deviation (it is defined in [21]). The main idea is to make the sender wait neither too long nor too short. After deducing packet loss by expiration of RTO, TCP enters the time-out phase. In the context of this thesis, detecting loss in such a way will be referred to as "time-out loss events" or just "time-out".

The TCP sender starts with an initial congestion window size of 1 packet (a further revision of the TCP specification allowed a larger number of packets in the initial window with an upper bound of roughly 4KB [8]; the "initial window size" refers to the window size after the three-way handshake), and it is in the slow start phase having an exponential growth of the congestion window. This behavior is kept till the window size reaches a certain threshold (*ssthresh*) or until a loss is detected. Once *ssthresh* is reached, TCP switches from slow start to the congestion avoidance phase. As already mentioned, loss can be detected in two ways, and the TCP window size is changed accordingly. In case three duplicate ACKs are received, TCP enters fast retransmission, which represents retransmission of a missing packet, and TCP congestion control switches to the fast recovery phase: TCP halves its congestion window size and its *ssthresh* value. Then TCP continues the window increase in the congestion avoidance phase. On the other hand, if the retransmission timer expires, the TCP sender assumes that the whole window of data is lost and the network is heavily congested, and therefore the congestion window is reduced to 1 packet, *ssthresh* is set to half of the window size before this reduction, and the TCP sender continues operating in the slow start mode.

Since other variants of TCP will later be referred to, i.e., TCP Tahoe, TCP NewReno and TCP Sack, the main distinction between them is provided here. TCP Tahoe [21] is an older version of TCP Reno and its characteristic is the non-existence of fast recovery. Consequently the window size is reduced just by retransmission time-out. TCP Tahoe is more vulnerable and the number of time-outs and therefore the number of packet retransmissions can be significantly greater than with the improved version — TCP Reno.

TCP NewReno [52] is a minor improvement of TCP Reno. This variant is sometimes able to cope with multiple lost packets from a single window more efficiently. For each missing packet belonging to the same window, starting from the second one, TCP NewReno waits for only one instead of three duplicate ACKs before retransmitting it, and therefore the fast retransmission phase is shorter.

TCP Sack [94] is an improvement of TCP NewReno achieved by introducing selective acknowledgements (Sack). With this feature, loss of multiple packets can be conveyed by a single ACK; thereby fast recovery/fast retransmission becomes shorter. In case of multiple losses in a single window, TCP NewReno will retransmit just one packet each round-trip time, but using Sack, the sender is informed by a single ACK about a number of lost packets which can be retransmitted immediately.

TCP NewReno and TCP Sack are just minor deviations from the TCP Reno version; on the other hand, the transition from TCP Tahoe to TCP Reno has significantly changed the dynamics of TCP congestion control and the modeling of these two versions notably differs.

Having given the most important aspects of the protocol we end the short description of the TCP protocol here; some additional details are to be seen throughout the entire thesis.

2.2 Queue management

An important factor that influences the performance of TCP and the entire network are queuing mechanisms. A queueing mechanism is responsible for managing the length of a queue, e.g. by dropping packets. These mechanisms can be divided into two groups [62]:

- **Passive Queue Management (PQM)**
- **Active Queue Management (AQM)**

Passive Queue Management — such a queue drops packets only when the buffer overflows and no preventive dropping is employed.

The best known passive queue management scheme is the DropTail queuing mechanism. It is the simplest scheme; it was the first scheme deployed in the Internet and a great amount of queues in the network still use this mechanism. All packets, arriving at a queue which is already full, are dropped.

Lengths of queues influence the performance of the network greatly. Larger buffers give a higher throughput but, at the same time, are responsible for a longer queuing delay which is for some traffic especially harmful (e.g. real-life applications). To be able to accommodate packets sent in bursts (e.g. due to the way TCP operates) and to produce as little as possible queuing delay, the length of a queue should be equal to the bandwidth $\times$ delay product (here a delay is the round-trip time). In a network with traffic mainly consisting of TCP flows, such a queuing mechanism can cause some additional problems, e.g., just a few connections occupy an entire buffer and other connections are locked out (phase effects); since the packets are dropped only when a buffer is full, this causes a buffer to be full for a longer time which introduces constant long queuing delays. Such a queue can produce global synchronization which means that all connections always experience loss at the same time.

There are some other variants of passive queue management, like Drop-from-Front PQM [81]. In case a packet arrives at a full queue, this scheme will drop the first packet from the front and accept the newly arrived packet. In this way, a sender is informed about congestion earlier (a congestion signal will arrive earlier comparing to a DropTail queue; the time difference is the time a packet would spend in a full queue).

Push-out [114] is yet another PQM scheme. Except being more demanding in the sense of work needed on a router, this scheme does not differ much from the DropTail mechanism considering

the network performance. When a packet arrives at a Push-out queue which is full, the last packet in the queue is dropped and the newly arriving packet is enqueued.

Active Queue Management schemes have been developed as a response to the problems seen with the simple PQM schemes. These mechanisms try to prevent growing of a buffer and queue overflowing by sending congestion signals earlier. In that way, the buffer is kept small and queuing delay is short. Such mechanisms usually drop packets randomly, which helps in preventing connections lock-out.

Random Early Detection (RED) [53] is the best-known AQM scheme and it is recommended by the IETF [20]. This algorithm drops packets with a certain probability before a queue overflows. This probability depends on the average queue length which is calculated using an exponential weighted moving average (EWMA) method — $avg(t) = (1 - w)avg(t) + wq(t)$, where $avg(t)$ is the average queue length, $q(t)$ is an instantaneous queue length and w is a weight parameter. Three additional parameters (min_{th}, max_{th} and max_p) influence the probability of packet discarding. As a new packet reaches a queue, the average queue length is calculated. If the average queue length is smaller than min_{th}, no packet is dropped; if $avg(t)$ is between min_t and max_{th}, the drop probability is calculated as: $p = max_p(avg(t) - min_{th})/(max_{th} - min_{th})$; for the average queue lengths being larger than max_{th}, packets are dropped with a probability of 100%.

Setting parameters of the RED algorithm is a challenging problem. The parameter w should be much smaller than 1; and its value controls the responsiveness of the algorithm (for a larger value of w, the instant queue length has a larger influence and the algorithm reacts on short-term bursts and short-term congestion; for a smaller value, a single state of congestion has an influence on the dropping rate at a queue for a longer time). The other parameters are also hard to tune, and they depend on the number of connections traversing a router as well as the round-trip time of connections among other things [62]. The current advice for setting the RED parameters [3] is to use an automatic configuration that is implemented and can be activated by setting "q_weight_" to -1, "thresh_" to 0 and "maxthresh_" to 0; the automatic configuration sets RED parameters, so that the target average delay in seconds, "targetdelay_", is achieved ("q_weight_", "thresh_", "maxthresh_" and "targetdelay_" are variables from the RED implementation in the ns-2 simulator. They correspond to w, min_{th} and max_{th} from the text above). As additional problems of RED instability were detected [44], the "gentle mode" was implemented which changes the dropping probability just in the range of a queue size between max_{th} and two times max_{th}. This mode has a packet discarding probability which linearly increases from max_p to 1 as the average queue length changes between max_{th} and two times max_{th}, and this makes the RED algorithm more robust to the settings of parameters "maxthresh" and "max_p".

In some network conditions, problems of the RED mechanism can be seen, e.g. high delay and jitter, unfairness (especially unfairness between TCP and unresponsive flows), and low throughput. Therefore other variants of the RED algorithm, that overcome these problems, are developed. They differ in the dropping probability function and queue length measure they use (whether they use e.g., the instantaneous queue length, the average queue length or if the dropping rate is calculated depending on per-flow measurements). Some variants are

Stabilized Random Early Drop [102] (uses instantaneous queue size and dropping function is a step function with three ranges), Double Slope RED [129] (also uses the average queue length but the function is changed and it has two slopes instead of one) and Random Exponential Marking [13] (the dropping rate is changed exponentially). Further, to solve the fairness issue, there are RED variants that calculate a per-connection dropping probability, e.g. Fair RED [84], Fair buffering RED [73], XRED [66], Class-Based Threshold RED [104] and Balanced RED [12]. Further examples can be found in [62].

Active Queue Management schemes can also implement Explicit Congestion Notification (ECN) [106] [107]. When this mechanism is used, a queue, instead of dropping a packet, sets a bit in the IP header; this is set in packets traversing a path from a sender to a receiver. Further this information is sent back to the sender as a bit in the TCP header. Upon receiving this bit, the sender behaves as if the packet had been dropped. Using this mechanism, packet drops are not eliminated completely and they are to be seen in case of a queue overflow.

2.3 TCP modeling

Since most of the traffic in the Internet uses TCP as a transport layer protocol, trying to understand and predict the performance of the network led to a great amount of effort being put into TCP modeling. The models range from very simple to very complex ones and many different mathematical approaches as well as many different assumptions about the network are used. Here an overview of these models is given.

A simple categorization of the existing modeling work would be:

- **models of a single flow**
- **models of multiple flows.**

From another prospective, also important for this thesis, a different grouping is possible — classifying models depending on assumed knowledge about the network. Therefrom we have the following two groups:

- models assuming the characteristics and the condition of the underlying IP network seen from the perspective of the TCP sender are known, e.g., the round-trip time of a connection, the loss probability experience by a connection etc. For these models the main goal is to estimate the throughput of TCP under certain network conditions, but mainly just the throughput and the transfer latency of a single flow are modeled.
- models assuming that more details about the network structure are completely or partially known, where by details, the topology, the link capacity, link delays and other parameters

of such a kind are meant. These models try to capture the behavior of a network as a whole and to study the influence of certain new techniques or the interaction between different approaches, like: special queuing mechanisms, new protocols, interaction between protocols, the behavior of the network in some extreme cases etc.

These two classifications overlap to a certain degree: in the literature, models assuming the network is described from the TCP source viewpoint are exclusively models of a single flow, and models considering multiple flows also require a more detailed network description. To the best of our knowledge, only the models presented in this thesis do not comply with this rule. Here, a model of multiple flows is derived; the network is observed from a TCP source-centric point of view. Assuming the existence of arbitrary cross-traffic in the network and assuming the exact network topology to be unknown, the network is characterized just by the packet loss and the round-trip time of the examined flows. In the literature there is one more modeling approach that in some way does not match this overlap — models taking a queuing theory approach for analyzing TCP. By extracting parts of these models, a description of TCP flows considering the same knowledge about the network is obtained (more details are given in following text). To give a comprehensive overview of all existing mathematical representations of the TCP protocol, the most important models from both groups are illustrated here. This will give a motivation for the approach chosen for our modeling efforts.

The first of the above proposed classifications will be used, and the listing of single flow analytical models will be followed by analytical models considering multiple flows. This study concentrates only on the modeling of TCP Tahoe, TCP Reno, TCP NewReno and TCP Sack which are the most relevant TCP variants for this thesis; TCP Vegas, FAST TCP etc. [123] have quite different congestion control algorithms, and models of such protocols are omitted here.

2.3.1 Modeling of a single TCP connection

There is a set of modeling efforts concentrated on the behavior of a single flow that aim at obtaining a closed form expression of the throughput of a single TCP connection. These models assume that feedback from the network is known, where by feedback, the round-trip time and packet loss are meant.

The TCP flow behavior is characterized by the TCP congestion control algorithm and the interaction between this flow and existing cross-traffic in the network is seen as packet loss. Therefore a model of a single TCP flow can be described by outlining two main characteristics: the mathematical approach taken for describing the TCP congestion control algorithm and the taken assumption about the loss distribution which is the main factor that determines the accuracy of the model.

A deterministic time between loss events is assumed in [95, 80]. The authors of [95] derive what is probably the most used TCP throughput equation. This simple model considers just the

congestion avoidance phase and takes the time between loss events to be periodic. This TCP throughput formula shows a rough dependence between the steady-state throughput (B) of a TCP flow and the loss probability (p):

$$B = \frac{MSS}{RTT}\frac{C}{\sqrt{p}}, \quad C = \sqrt{\frac{3}{2}} \tag{2.1}$$

where MSS is the segment size and RTT the round-trip time. As denoted in [95], a similar equation has been published in the literature even before [45, 79], although with a different constant C. These studies are in line because the value of this constant can differ depending on the TCP variance observed (e.g., use of delayed ACK, assumed loss distribution).

The second, very often cited TCP throughput equation is presented in [103]. It can be shown that the assumption about the loss distribution gives a deterministic time between congestion events [9]. This equation is quite accurate and at the same time fairly simple, and therefore it is chosen as a base for the model developed in this thesis. For a better understanding of our model, a more detailed description of this equation will be given in Chapter 3. The time-out phase is included, and the throughput of a single TCP flow, as proposed in this paper, can be calculated by:

$$B = \frac{MSS}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \tag{2.2}$$

MSS, RTT, p having the same meaning as above, T_0 being the TCP retransmission time-out value and b the number of packets acknowledged by an ACK.

Further this model is extended in [27] by a precise description of the connection establishment phase and an analysis of the initial slow start (modeling exactly how the window size is increased). This work mainly focuses on calculating the latency of a finite TCP flow and it is further enhanced in [116] by assuming that time-outs and the slow start phase can occur any time during the connection and each slow start phase is modeled more precisely then in [103] (in [103], the exponential growth in the slow start phase is not modeled and the slow start is represented just by the window being decreased to 1 packet). The authors of [116] also obtain a steady state throughput equation of a TCP flow. This closed form is slightly more complex than the one derived in [103], although numerical results as well as measurements presented in [116] show that these two formulas are almost the same. This is an expected result since the slows start phase, which the authors introduce, does not have a high impact on long flows (the number of packets of a longer flow sent in the slow start phase, between the time-out detection and the point at which a flow reaches $ssthresh$, is significantly smaller than the number of packets sent in the congestion avoidance phase; in case a higher loss rate experienced by a long flow, the time-out phase can occur more often but then the window size is fairly small and the difference between the window increase in this two phases of the congestion control algorithm is rather negligible).

In [98] a TCP flow that traverses one bottleneck router is considered. The authors of [98] use stochastic differential equations to derive a fluid model of a TCP flow. They obtain a closed formula of the throughput of a single TCP flow with the assumption that the network sends loss signals to the sender of the flow with a Poisson distribution. This work neglects the *slowstart* phase of the congestion control algorithm. The TCP model obtained in this work can be reduced to the equation from [95] under the assumption that there is no limit imposed on the TCP window size.

All these models take some concrete assumption about the distribution of packet loss, although the loss in the Internet can have a general distribution, ranging from a non correlated distribution with a deterministic time between loss events to a fairly burst distribution. Therefore the work presented in [9] tries to derive the throughput of a TCP flow by taking as few assumption as possible about the loss distribution, only considering the loss to be some stationary exogenous random process. For a concrete network, this model gives the possibility of choosing the appropriate model for congestion events. This model presents the change of the window size as a continuous function (so-called fluid model) which makes the mathematical analysis easier by enabling the use of integration and differentiation. On the other hand, all fluid models introduce an error by neglecting the discrete increase and the authors of [9] describe a possibility for correcting this error.

The authors of [75] use the Markov renewal-reward processes. The number of states in the Markov chain depends on the advertised receiver window, and with its increase the complexity of the solution will grow.

Here we are ending this subsection. The most important examples of single flow's models are presented. Some additional work is to be found in literature but, since their are relatively old and some models presented in this subsections are their extensions, we omit them here.

2.3.2 Models of multiple TCP connections

As already stated, models considering multiple flows are always coupled with analyzing a network as a complete system. Performance analysis of a large IP network is an important aspect of network research. This research area studies the behavior of the network as a whole and provides tools for analyzing the performance and, above all, the stability of new approaches before their deployment in a real network. Network simulators, like the ns-2 simulator [4], enable the representation of packet level interactions in a network. Considering the size of the Internet, simulators are not able to cope with its growth, and network analysis of a larger network which size is still far from a size comparable to the Internet are rather limited by the required execution time. Therefore recent analytical models of IP networks have received growing attention. They are able to capture the behavior of an arbitrarily large network, where usually the complexity of a model does not increase with the network size as in the case of the simulators. A further advantage over simulators is that they are also capable of analyzing some theoretical situations. For example, in [17], the behavior of an IP network when the number of TCP connections tends to infinity is studied. Using modeling, often, limitations of a current

system are shown and possible directions for improving the system are hinted at — for example, the analysis presented in [44] showed the instability of the RED queue with an increase of the number of flows traversing that queue.

Since Internet traffic mainly consists of TCP flows, the modeling of TCP plays an important part of each network model. The TCP protocol modeling is a challenging problem because of its algorithm complexity and especially its non-trivial interaction with the underlying IP network.

There is a vast amount of work concentrated on the modeling of IP networks and a variety of mathematical approaches are used: modeling flows as fluids (so-called fluid model), queueing networks, observing a network as an optimization problem etc. Only some of the most relevant efforts will be presented here, as the complete overview of all TCP models would require a separate study.

Fluid model

Over the years fluid models presented in the literature have been gradually improved. The first fluid models started with the modeling of TCP elephants (long-lived TCP connections) and assuming just a single bottleneck link [98, 99]. This work has been further extended to accommodate the modeling of TCP mice (short-lived TCP connections), multiple paths and bottleneck links (like [86, 6, 17]).

A fluid model of a single flow [98], described in the previous subsection, is extended in [99]. The authors of [99] consider a set of TCP flows traversing queues with an active queue management schema (AQM). In [98], the loss is considered to have a Poisson distribution and to be independent of the TCP sending rate; on the other hand, in [99] a complete, closed system is modeled (assuming the modeled flows to be the only existing flows in a network with multiple congested routers). Jump process driven stochastic differential equations are used and as the result a system of ordinary equations is obtained that can be solved numerically and the throughput of TCP flows as well as the queues' length and round-trip time can be obtained. This model is developed to facilitate the analysis of interaction between the congestion control mechanism and AQM schemas. Further this model is used to analyze the RED queuing mechanism in [53]. The drawback of this model is the fact that the order in which a flow passed through the router is neglected, therefore leading to fairly incorrect results in some cases (e.g., a simple case would be: when flows are traversing two links with the same capacity, the model would not capture the traffic shaping at the first link and would assume the same traffic arrives at both links) and the *slowstart* phase is omitted for simplicity. Authors in [86] improve this model by confronting these problems. As in the previous model, the average windows size (throughput) of each flow class (flows of a class traverse the same set of routers) is obtained. They also add an unresponsive background traffic by using traffic models defined in [64], instead of generating individual flows. Nevertheless the assumptions taken for this model are still not identical to the requirement considered for our modeling work because the network topology needs to be known.

The model presented in [11] examines two TCP connections under different loss strategies. The authors consider a broad range of loss strategies: from non correlated — loss always affects just one flow, to the correlated — always both flows experience loss. It is tempting to believed that a simple extension of this model for the case of multiple flows that we are interested in, is possible. However the solution is not as simple as it seems and a more profound discussion of this issue will be given in Section 2.3.3 where the summary of all models and their relations to our modeling efforts are given.

Another challenging problem in the fluid based modeling is the DropTail queuing mechanism that introduces burst losses and synchronization between flows that are fairly hard to model. The authors of [6] incorporate a possibility that queues implement the DropTail mechanism and also extend the previous model by an important aspect — TCP mice. This model uses partial differential equations which allow a description of the TCP window distribution per flow class rather than just the average.

Fluid models only consider a longer-term deterministic flow-level traffic dynamic, neglecting short-term stochastic behavior of flows observed on the packet-level (packets of a window are sent in a burst rather than spread over round-trip time). Therefore the above presented fluid models are not capable of capturing network behavior in a case of underutilized links. This problem is discussed in [6] and a couple of proposed solutions how to incorporate randomness on the flow level as well as on the packet level are presented in [5, 29] obtaining hybrid fluid models — mixing fluid representation of flows and packet level and flow level randomization.

The modeling approach from [17] takes a differential equation representation of TCP flows and assumes that queues use the RED mechanism. For solving the model a mean-field approximation of the entire network system is used. Further on in [16] also the complete network with multiple queues is considered and the outcome shows that the interaction between TCP flows and dynamics of the system with small buffers can be described as billiards in the Euclidean space.

There are models that use a presentation of TCP flows as fluids but do not use differential equations, like [15] which gives a model of multiple TCP flows that traverse a DropTail link. The proposed model gives the throughput of TCP flows as products of random matrices and in addition, an investigation of the fairness of TCP flows under different synchronization levels is given.

A model of multiple flows sharing a single bottleneck link is presented [10]. This derivation is based on [15] and extends this work by accounting for a finite buffer size. The outcome is a simple formula that identifies dependency of the throughput of n TCP flows on the capacity of the bottleneck link and a value of n and it is shown that already 3 connections cause 90% link utilization. The authors of this work also come to the conclusion that an aggregate throughput achieved by the connection is insensitive to the loss distribution, on the other hand, the steady-state distribution of the throughput is influenced by the loss distribution.

Queuing network modeling

Queuing theory is another approach to the modeling of a complete networking system. In this case, the model of a network system (considering the network system as a set of routers, links and TCP sources) is separated into two parts:

- **TCP sub-model** — modeling the behavior of a TCP source or TCP sources, thereby modeling the TCP congestion control algorithm.
- **Network sub-model** — presenting the network as a set of routers and links.

The TCP sub-model and the network sub-model are presented and analyzed using queuing networks. The TCP congestion algorithm is mainly represented as a set of states that are designated by the window size and the current TCP congestion control phase (the slow start or congestion avoidance phase). Minor deviations from this TCP sub-model can be found in some publications. The state transitions of the TCP sub-model network give a very precise image of the TCP window size growth prescribed by the TCP congestion control algorithm. Each queue of the network sub-model is a node of the modeled network. The proposals in this area also differ in the complexity of the chosen network topology, or rather the complexity of a topology that can be represented.

Further these models define interaction between these two parts. The solution of the complete system is gained by performing a sequence of successive refinements of these two parts. The outcome of the TCP sub-model is used for tuning the network sub-model and vice versa. The alternating tuning of these sub-models is carried out until further analysis produces just minor, negligible improvements. This iterative process gives a fairly non-trivial calculation complexity.

To the best of our knowledge the model presented in [30] is the first model of multiple flows using the queuing theory. The authors of [30] start by defining an analytical model of each of the above mentioned parts, assuming that they are statistically independent. The TCP sub-model part differs from the general approach presented above in that the evolution of the TCP window size (just TCP Tahoe version is considered) is combined with the application behavior. An ON/OFF behavior of the application is assumed, and it is modeled by two states: the inactive and the active state. To reduce the complexity of the model, TCP sources are grouped into classes where each class consists of flows which traverse the same path and have the same application behavior and the same propagation delay. All flows belonging to a class are represented by the same queuing network model and they have the same average rate (Γ). The outcome of the TCP sub-model is the aggregate rate of the packet generation of all sources (denoted by $\lambda = \sum_{i=1}^{n} N_i \Gamma_i$, where n is the number of classes and N_i the number of flows in class i). The authors assume the network to have a single bottleneck link modeled by an M/D/1/B queue and the input traffic to be a Poisson process with parameter λ (the real distribution of the traffic produced by the TCP sub-model is neglected). The model of the network produces the loss probability and delay that are used for a refinement of the TCP sources model. Further, an example of a network with two nodes is presented.

The TCP congestion control algorithm is described using a closed queuing network in [57]. This work is based on a theoretical investigation of a possible representation of an arbitrary congestion control using closed queuing networks, described in [35]. N TCP connections are presented using a closed network of M/M/∞ queues (Q), where the states are defined as described above (we refer here to the general case). Each TCP connection can be in one of the states, and N_Q is the number of connections in a state. The input into this closed queuing network is the packet loss probability and the round-trip time which are calculated in the network part of the model. Routers in the network are modeled with M/M/1/B queues (assuming the presence of a significant number of DropTail queues in the network). Using this model, fairly more complex networks, compared to the previous model, can be represented.

A further extension of this basic model can be found in [56]. Here the queuing network of TCP states is adapted to accommodate the possibility of arrival and departure of connections and an adequate, in this case, open multiclass queuing network was used. In this way short-lived TCP connections could be considered too.

Modeling in Max-Plus algebra and as an optimization problem

This overview of the models of multiple flows will be ended with a representation of TCP in max-plus algebra and as an optimization problem.

A TCP packet level model, using the max-plus algebra, is proposed in [14]. The model can present a network with no background traffic as well as a network in which the observed flows are not the only flows in the network. The influence of the background traffic is presented as random loss in addition to loss caused by buffer overflows. This model can be of interest for the application presented in this thesis but the complexity of this calculation limits its usage ($2n(K{w_{max}}^2) - n$ is the number of packets traversing K routers and w_{max} is the maximal TCP window size).

Interpreting the Internet as an optimization problem that can be solved with a distributive algorithm is the subject of the work presented in [70]. The distributed algorithm consists of two parts: the network part and the user part, which are shown to solve the global optimization problem [70]. The network optimization problem can be presented as the primal and dual problem. Respectively, interaction between the user part and the network part can be interpreted in two ways: as the congestion indication feedback or as the explicit rates calculated depending on the charge a user is willing to pay. The author of [70] shows the usefulness of a protocol that enables weighted proportional fairness between flows to be achieved. If each user chooses weights in such a way that its own utility is maximized, the global optimum is achieved. There is a vast amount of further research efforts taking this approach, e.g., [88, 76, 87, 97].

2.3.3 Summary

As shown above, there are many attempts to capture the behavior of a TCP flow as well as of multiple TCP flows taking a broad range of mathematical approaches. To the best of our

knowledge a model of multiple TCP flows with the assumption that the network is observed from the TCP sources' perspective and characterized by the packet loss probability and the round-trip time will be first presented here.

The models described in Section 2.3.2 assume that the network topology and the complete traffic that traverses a given network are known and therefore it would not be possible to use them if arbitrary cross-traffic exists in the network. For most of the models from this group, the obtained final results are rather complex or they include numerical solving of differential equations or iterative tuning of two separate systems. This constrains their applicability. There are two models in this group that in some way consider incorporating the background traffic into the network model. The authors of [86] assume a certain rate and a certain rate distribution of the background traffic on each queue in the network, thereby increasing the complexity of the presented model. As already stated, the network topology needs to be known too and the solution is obtained numerically which is rather impractical. The second model in this group that enables a broader definition of the traffic existing in a modeled network is proposed in [14]. Again, the mathematical approach chosen in this case does not produce a closed form solution that could be widely applied and easily used. This model assumes that the network topology is known and whether an approximation of the network topology by a single bottleneck will yield accurate results is not certain. Even this simplification leaves us with an additional unknown — the bottleneck link capacity. Obtaining it would require additional measurements. The authors in [10] provide a very simple closed form equation for the throughput of parallel TCP flows. As with most other models in this group, the existence of cross-traffic is not assumed and a simple way of incorporating this aspect in the calculation is not known.

The queuing theory models always separate modeling into a TCP sub-model and a network sub-model, and the system is solved iteratively with only the loss rate and the round-trip time fed into the TCP sub-model. Hence extracting just the TCP sub-model could be a promising approach. Each window size that a TCP sender can achieve need to be represented by a queue in the TCP sub-model; The window can have any size between 1 packet and the advertised receiver window. Consequently the complexity of a queuing network that describes TCP flows, in other words, the number of queues needed in the queuing network, depends on the size of the advertised receiver window and grows with its increase. For example in a case of the maximum segment size being 1024 bytes, the receiver window having a size of 64 Kbytes and the number of successive time-outs before a connection is closed being 16 (these are the values used for validation in [57]), the maximum window size is 64 and the number of queues needed is 357. Solving such a system has a complexity of $O({M_Q}^2)$ where M_Q is the number of queues in the queuing network. Therefore such a model would not produce a practical throughput equation.

Fluid network models are mostly used for studying certain aspects of interaction between TCP flows and the underlying network implementing certain queuing mechanisms. All these models assume that the observed flows are the only flows in the network, and by interacting between each other and with implemented queuing mechanisms, loss is produced in other words, a complete, closed system is modeled. To simplify matters, some of these models omit the possibility of time-out loss indication (like [11, 86, 99, 64]). Others are more detailed (they include time-outs), like the models presented in [5, 29]. On a first thought such a model could be simplified and used as the model that we are aiming at. For example, in [11], a model of two TCP con-

nections is obtained and it is tempting to believe that a simple extension is possible for the case of multiple flows which we are interested in. A study of different loss strategies is given, e.g., loss always affects both flows, just the flow with a higher rate is affected, or a random flow is affected; but which strategy should be chosen to match the loss distribution in a network with arbitrary cross-traffic is not a subject of this work. This is an important, maybe even crucial part of a modeling work that has the goal declared in this thesis and it requires a certainly non-trivial study.

The closed form solution of the throughput of TCP is obtained in the models illustrated in Subsection 2.3.1, but unfortunately these efforts assume a single flow. A straightforward idea for applying these formulas to the case of parallel TCP flows would be a simple multiplication by the number of flows, but the packet loss rate of multiple flows measured by observing them as a single cumulative flow, which is a more practical way of measuring loss as discussed in Section 3.1, yields a result that is smaller than the loss rate of individual flows. Therefore such an extension will always overestimate the throughput of the parallel flows (for further discussion the reader is referred to Section 3.1). As shown in this thesis, a non-trivial extension of such a model is required.

For the modeling work presented here a fluid or discrete modeling approach could be chosen. In case of the former technique some aspects need special attention. Apart from an error introduced by neglecting the discrete increase of the window, a more important uncertainty exists: it is not obvious whether, using a fluid model approach, is possible to obtain a closed formula, considering that these models usually involve differential equations, partial differential equations etc.

The most widely used method for calculating the throughput of a single TCP flow under certain network conditions is described in [103]. It seems that it achieves just the right balance between precision and ease of use. Hence this model is chosen as the foundation for deriving the formula for calculating the throughput of parallel TCP connections which will be presented in this thesis.

2.4 “TCP-friendly” protocols

It is easy to show that non-responding traffic, like certain UDP traffic, can cause harm to well behaving one. Therefore, for some time TCP-friendliness was a focus of research efforts. A TCP-friendly protocol is a protocol that does not consume more bandwidth than the standard TCP protocol would under the same network conditions. Some research outcomes in the direction of TCP-friendly protocol are: Rate Adaptive Protocol (RAP) [109], TCP Emulation at Receiver (TEAR) [112], Binomial Congestion Control [19] and TCP-friendly Rate Control (TFRC) [50].

We will focus on TFRC which is the most relevant one for the work presented here. A part of our work extends this protocol.

TCP-friendly Rate Control is a rate based protocol and it makes use of the equation presented in [103]. In [103], an equation that calculates the steady-state throughput of a TCP flow is presented; some more details are provided in Subsection 2.3.1 and the formula is given by equation (2.2). The throughput is calculated depending on the loss rate (p) and the round-trip time (RTT) on a path as well as the duration of the retransmission time-out value (T_0) and b (the number of packets acknowledged by an ACK). In TFRC, the b parameter is set depending on the TCP variant whose throughput TFRC wants to achieve; the loss rate and the round-trip time are measured. T_0 is calculated using the RTT value, and two calculation methods are possible: the standard TCP method described in [21] and using the value of $4RTT$. Packet loss is measured on the receiver side and conveyed to the sender over receiver feedback packets. The feedback packets are also necessary for a round-trip time measurement. To be able to get the most recent sample, these packets are sent at least once per round-trip time (if the sender is sending less than one packet per round-trip time, a feedback is sent for each received packet). If a loss or ECN mark is detected the receiver sends a feedback packet without waiting for the current round-trip time period to end. After measuring the necessary parameters, a TFRC sender calculates the throughput using the equation and sets this value to be the sending rate. In this way TCP-friendliness is ensured. For all calculations and measurements, the exponentially weighted moving average of round-trip time samples is used (the recommended weight parameter is 0.9). Packets containing receiver reports can be lost too. If no receiver feedback has arrived for several round-trip times, a TFRC sender starts reducing its sending rate until it stops sending. This waiting time is set to $4 * RTT$.

The primal purpose of the TFRC protocol is to provide a responsive network service for multimedia traffic. Considering the characteristics of the real-life applications, i.e. their need for a smoother rate, TFRC uses an average of the measured parameters calculated over some period of time instead of instantaneous values.

The packet loss rate is measured by accounting the number of packets sent between two loss event indications (such a period is denoted as a loss interval). A loss event indicator is a lost packet that forces TCP to reduce its sending rate; the occurrence of such an indicator is called a "loss event". Since TCP reduces its rate at most once in a round-trip time, only the first lost packet of a number of consecutive losses that are less than a round-trip time apart is a loss event indicator. Like TCP, TFRC ignores all lost packets that are less than a round-trip time away from a loss event indicator. If a loss of packet l is detected at time t_l and packet l is a loss event indicator, packet loss detected at time t_n which is $t_n < t_l + RTT$ is considered to be a part of the same loss event and cannot start a new loss interval (also here, the exponentially weighted moving average of round-trip time samples is used). For achieving a smoother rate, the k most recent loss intervals are considered and their exponential weighted moving average (EWMA) is calculated. In [50], the value of k is set to 8. We denote by I_i, $i = 1..8$, the number of packets sent in the last k loss intervals and by I_0 the last interval that still has not been ended by a loss event. The average is calculated using:

$$I\_wa = \frac{\sum_{i=1}^{k} I\_i \;\; w\_i}{\sum_{i=1}^{k} w\_i} \tag{2.3}$$

and the weights are: 1, 1, 1, 1, 0.8, 0.6, 0.4 and 0.2. Since the most recent loss interval is still not complete, it is included in the loss rate calculation (in other words, the loss rate is

calculated using intervals I_0 to I_7) if, in this way, a larger value is obtained. Then, the loss rate is $1/I\_wa$.

Before the first loss occurs, TFRC does not have any information about the network and during this time, it emulates the behavior of the standard TCP protocol. In other words, it doubles its rate every round-trip time. The first loss interval could not even be used as a valid sample for loss rate measurements because, till the first loss, the rate of a flow has not been controlled by the equation or it has not had the TCP steady-state behavior but it has had an exponential growth; when the first loss occurs it is assumed that the current rate is the double of the available bandwidth on a path (since the sending rate is doubled each round-trip time) and the sending rate is set to the half of the current rate.

As simulations and tests using the "Dummynet" network emulator have shown [50], TFRC does not harm TCP traffic in a wide range of network conditions and at the same time obtains a smoother rate.

2.5 "Non-TCP-friendly" protocols

Since over the years the TCP protocol has succeeded in preserving the stability of the Internet, the requirement that all flows have a conforming behavior compared to the one of the standard TCP protocol became a must — in other words, protocols need to be TCP-friendly. A TCP-friendly protocol is a protocol that does not consume more bandwidth than the standard TCP protocol would under the same network conditions. But as soon as TCP had hit its limits by not being aggressive enough for existing link capacities (i.e. being incapable of saturating the Internet links), a great amount of research efforts has been put into developing protocols that overcome this drawback. Some work has also been motivated by the fact that the fairness offered by the TCP protocol does not reflect users' perspectives; the proposed protocols have implemented some kind of weighted congestion control. Regardless of the motivation, "better-than best effort" protocols can be split into two groups:

- **congestion control with a higher "aggression"** — protocols belonging to this group are more aggressively probing for the available bandwidth than TCP, although some of them show this property just in case of a non-congested network. A common mark for all protocols belonging to this group is that the aggression cannot be characterized as a multiple of the aggression of the standard TCP. Such protocols are: BIC [128], CUBIC [110], FAST [68, 122], Scalable TCP [71] Compound TCP [119], HighSpeed TCP [47], and many others. As the behavior of these protocols greatly diverges from that of the MulTFRC protocol, a detailed overview of these protocols is omitted here.
- **weighted congestion control** — the aim of these protocols is to emulate the behavior of multiple TCP flows; they can be characterized as "n-TCP-friendly" just like the protocol presented in this thesis (MulTFRC), and hence a comprehensive overview of these protocol is required.

This classification is only one possible example and it is chosen since it provides a view that focuses on an aspect of interest in this thesis: characterizing protocols by their TCP-friendliness. Many other classifications are possible, for example the author of [23] argues that flow fairness does not coincide with any economical form of fairness and that a discussion about TCP-friendliness is unnecessary; it is rather fairness between economical entities that is important. As he suggests, the users (in this case economical entities) should be able to allocate rates to their flows depending on the benefit they receive for them and not to be restricted to a single possibility — TCP-friendly protocols. Different applications should receive different rates because from the users' perspective the importance of each application is not the same. For this study the classification above focuses on a fairly unimportant aspect.

Concerning aggression, the other extreme are protocols offering a "less-than-best effort" service, i.e., protocols that are less aggressive than the standard TCP protocol. Since, as already mentioned, the flow aggression of the MulTFRC protocol can be any real number including values less than one, MulTFRC can offer such a service too. Consequently any proposal that enables the rate of a transfer to be less aggressive than that of the standard TCP protocol is of interest too.

Weighted congestion control

Since multiple TCP connections are simply faster than a single one, the simplest way for achieving a faster transfer, without introducing a new congestion control algorithm, would be a flow that sends data over multiple connections, e.g., GridFTP [7] and *bbcp*[1]. GridFTP is an extension of FTP which utilizes multiple TCP connections in a way where a file to be transferred is streamed over more than one connection. Such methods have several disadvantages. Data transfers must be multiplexed onto individual TCP flows (e.g., by offering large files which are split into several parts for easier parallel downloading). Parallel flows increase overhead, as connection state information must be kept for each flow. Moreover, the granularity of aggression is limited, leaving only the possibility that a transfer is "n-TCP-friendly" with n being a positive integer.

Another approach to achieve "n-TCP-friendliness", and also a more elegant one, would be the development of a transport layer protocol that gives the behavior of a flow comparable to the one of n standard TCP flows. There are a couple of examples that conform with this description.

To the best of our knowledge, the MulTCP protocol [36] is the first attempt in this area. It imitates the behavior of n standard TCP flows by having an n times higher increase factor than the standard TCP protocol ($n/cwnd$) and decreasing the window size by $(n - 0.5)/n$ in case of congestion (it assumes that all flows have the same window size, therefore the window size of one flow is the n-th part of the cumulative window size, cwnd). This protocol reacts rather drastically upon time-outs by reducing its window size to 1 packet; on the other hand, n real TCP flows are probably less vulnerable to time-outs. The second problem of MulTCP is that the loss rate normally experienced by a MulTCP flow is smaller than the loss rate of n standard TCP flows [77], making MulTCP too aggressive. This causes MulTCP to have a sending rate

[1] http://www.slac.stanford.edu/ abh/bbcp/

comparable to that of n TCP flows just for a limited range of the aggression value, as shown in [36] and also in Section 4.2.4 where, because of the similarity with MulTFRC, a comparison between these protocols using simulation will be presented.

Probe-Aided MulTCP (PA-MulTCP) [77] is based on MulTCP and it tries to overcome the last stated problem of MulTCP. It is doing so by using probes. The goal of these probes is to determine a loss rate which is closer to the loss rate experienced by a real TCP than the loss rate of the original MulTCP. 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. These probes and the second control loop introduce some overhead and increase the complexity to the protocol.

Stochastic TCP [59] is a different approach to achieve the behavior of a flow to be like that of multiple TCP flows. A Stochastic TCP flow has a single congestion window that is an aggregate of a number of virtual flows. The inflation and deflation of a virtual flow directly affects the congestion window of the aggregate. The selection of a virtual TCP stream to be increased or decreased is achieved by a stochastic method assuming all flows can be affected with the same probability. The authors of [59] also propose a way of reducing the aggression of multiple virtual flows by artificially increasing the round-trip time of these flows, more precisely by increasing the number of acknowledgements needed for a window increase.

MPAT [117] is another mechanism that is concerned with controlling aggregates. Unlike PA-MulTCP however, MPAT maintains the control loops of individual TCP flows and lets them share their congestion state. In an example that is given in [117], if two TCP flows would be allowed to send 5 packets each but the bandwidth should be apportioned according to a 4:1 ratio, MPAT could allow one flow to send 8 packets and let the other flow send 2. Such a differentiation between flows is a way to provide Quality of Service (QoS) via congestion control.

All the work mentioned so far, including MulTFRC and in particular schemes like MPAT which use congestion control as a building block for QoS, leverage the effect of n-TCP-friendliness. There is also some work where such behavior is regarded as harmful, and appropriate countermeasures are taken. The Congestion Manager (CM) [18], for example, provides a unified congestion control mechanism for multiple connections between two hosts in the network, thereby eliminating any potential fairness issues. Congestion control information is also shared between multiple flows in Ensemble-TCP [42], but without entirely dislocating the rate control, making it easier to implement than the Congestion Manager.

The Coordinated Control protocol (CP) [101] is the most relevant one for the work presented here. Therefore, it will be further investigated and also compared with the MulTFRC protocol later in this thesis (Section 4.2.4). It basically emulates the behavior of a number of TFRC flows by multiplying the equation used in TFRC by the number of flows. The solution is not as simple as that, since as shown in [101] and also discussed in Section 3.1, measuring the congestion events rate used in the standard TFRC yields an incorrect estimate in the case of multiple flows. Therefore a stochastic technique for calculating the loss event rate of a single virtual flow is applied.

2.5.1 "Less-than-best effort" protocols

On the extreme end of "friendliness" towards 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 [120], for example, is a variant of TCP Vegas [22] with a more conservative congestion control behavior. TCP Vegas tries to estimate the bottleneck queue length and linearly increases or linearly decreases its window size with the aim of keeping the queue rather small. TCP Vegas is developed with the idea of being able to compete with the standard TCP protocol; on the other hand, TCP Nice changes the TCP Vegas congestion control algorithm to make it suitable for the background traffic by applying a multiplicative, instead of linear, reduction of the window size in case the round-trip time increases.

Just like the previous two algorithms, TCP Low Priority (TCP-LP) [78] 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 the standard TCP. Unlike TCP Vegas and TCP Nice it uses one-way delay to avoid reacting upon delay fluctuations caused by cross-traffic on the reverse path.

4CP [85] is another window based protocol that provides a "less-than-best effort" service, but it takes a different approach. This protocol aims at providing the foreground traffic with a minimum sending rate by forcing a certain loss rate (p_{tar}). If the loss rate is p_{tar}, a TCP flow will have the desired minimum rate. Therefore, when the packet loss rate is higher than the target loss rate, the protocol forces the sending rate to be 0. Otherwise the protocol sets the sending rate for the background traffic in a way that will ensure that the target loss rate is achieved.

The above listed proposals for providing a "less-than-best effort" service are transport layer solutions, just like the one presented in this thesis, but such a behavior can be achieved in different ways too. Application layer mechanisms are suggested in [72] and [118]. In both the rate of a flow is controlled by manipulating the receiver window. And further possibilities are given: in [43], by generic idle-time scheduling strategy in an operating system, in [28], by using class based queueing mechanisms in routers etc. A complete list of "less-than-best effort" schemas can be found in [124].

2.5.2 Summary

Some of the proposed protocols seem to achieve the same goal as MulTFRC, therefore a comparison with our protocol is necessary. This group of protocols embraces: MulTCP, Stochastic TCP, and, of course, CP as the most related protocol. A detailed investigation of these protocols will be given in Subsection 2.5. We did not run simulations with PA-MulTCP because the original code was not available and reimplementing it did not seem to be possible merely based on the description in the paper.

Protocols and application support systems that open a couple of separate TCP connections (i.e. GridFTP and MAT) are not of a special interest to us since the behavior of these protocols and applications is comparable to that of multiple TCP connections.

Since increasing delay is an effect that can be measured earlier than packet loss or ECN marking [107] (which are the normal congestion indicators of standard TCP), relying on it is a common theme in a work aiming at providing "less-than-best effort" (i.e. TCP-LP, TCP Nice). This is not the case for MulTFRC with $0 < n < 1$. Therefore our protocol is likely to be more aggressive than these alternatives if they share a bottleneck at the same time; on the positive side, this aggression is tunable in MulTFRC. On the other hand, the 4CP protocol has a different target — to limit the impact of the background traffic on the important traffic, and it could be assumed that this protocol does not have a constant aggression; the aggression is changed as the number of other flows traversing the same bottleneck varies. Comparing it to "n-TCP-friendliness", 4CP would have $n = 0$ if the loss rate is greater than p_{tar}; otherwise n would be moving towards 0 as the number of flows in the network increases.

2.6 Fairness

In a system with multiple entities sharing a resource or a number of resources, fairness is an important issue. The Internet is a system of that kind — fairness between data flows implies that the flows should share the bandwidth of a bottleneck link in the case of congestion fairly. This is not the only possible way to look at fairness in the Internet, e.g., in [23], establishing fairness between economical entities, not between data flows, is suggested (this will be discussed later in this section). By not looking at the entities that are compared, but just at the way resources are shared between flows, we can have various forms of fairness: max-min fairness, proportional fairness, weighted proportional fairness etc.

Max-min fairness: Flows share the bottleneck link according to max-min fairness if an increase of the rate of any flow would cause the rate of some other flow that already has a smaller rate to decrease. A set of connections S is observed. $\vec{x}$ is the vector of rates of all connections, where x_s is the rate of connection s and $s \in S$ (the same notation will be used in further definitions). A more formal definition of max-min fairness is:

> A rate allocation $\vec{x}$ is max-min fair if for any other feasible rate distribution $\vec{y}$ and $y_s > x_s$ there must exist some s' such that $x_{s'} \leq x_s$ and $y_{s'} < x_{s'}$.

Therefore the flows will always share the bottleneck link equally, except in case a certain flow is incapable of using its allocated bandwidth and the rest of its share will be equally distributed between other flows. Max-min fairness gives absolute priority to flows with smaller rates in such a way that these flows can always send with the maximal rate. For a longer discussion about max-min fairness the reader is referred to [82].

Proportional fairness: A rate allocation is proportionally fair if any other feasible distribution of rates (feasible in a sense of given link constraints) yields the sum of proportional changes to be negative. In other words:

> A feasible distribution of rates $\vec{x}$ is proportionally fair if and only if any other feasible rate allocation $\vec{y}$ results in:
>
> $$\sum_{s=1}^{S} \frac{y_s - x_s}{x_s} \leq 0$$

According to this distribution, a smaller rate can be decreased by a certain factor if it implies an increase of another rate by a large factor (see [82] for more details). The TCP protocol with its congestion control (Additive-Increase, Multiplicative Decrease congestion control algorithms — AIMD), regardless of the influence of deployed queuing schemas, tends to distribute the rates of data flows according to proportional fairness [70], although this is only a rough approximation [121].

Weighted proportional fairness:

> Let $\vec{w}$ be a vector containing weights of connections, where w_s is the weight of connection s and $s \in S$. An allocation $\vec{x}$ is weighted proportionally fair if any other feasible allocation $\vec{y}$ yields:
>
> $$\sum_{s=1}^{S} w_s \frac{y_s - x_s}{x_s} \leq 0$$

In this case a connection with the weight of two will get the same rate as two connections with a weight of one. Weighted proportional fairness is studied in [70]. Considering weights to be the price each user is willing to pay per unit time, it is shown that an allocation of rates conforming to weighted proportional fairness, where weights are chosen so that the utility of each individual user is maximized, results in the global optimum, i.e., the network utility is maximized [70]. Motivated by this study, the authors of [36] present a system that uses MulTCP, a protocol which can be used to realize weighted proportional fairness: connections are associated with a price, and the amount that users would pay for their rate allocations is proportionally fair. This kind of fairness can be a basis for service differentiation in the Internet. MulTCP achieves weighted proportional fairness by acting like a number of standard TCPs at the same time. Being as aggressive as a number of "TCP-friendly" flows, the MulTFRC protocol also provides this kind of fairness.

Fairness has been a matter of intense debate in the network community and still is an unsolved issue. Therefore, without starting a discussion about pro and contra of each separate scheme, a historical overview of fairness criteria recommended and proposed for the Internet as well as current tendencies will be given.

Methods for sharing network resources have been an area of interest starting from early days. A straightforward recommendation of an equal sharing of the bottleneck link was suggested in [108]. Further on, in [100], the use of a per-source fair queuing at routers was proposed, although the threat of possible malicious hosts that could use multiple addresses was recognized. In 1988 the TCP congestion control was proposed [67] and deployed. From that point on, just by using the TCP protocol, a certain form of fairness was achieved. The TCP congestion control algorithm offers an end-to-end flow fairness which is "window fairness" [34], i.e., the same window size is allocated for each flow independent of its round-trip time and packet size; therefore the TCP protocol shows bias against flows with a larger round-trip time [82]. TCP flows share the bottleneck link approximately proportionally fair [121].

In 1997 the term "TCP-friendliness" was created [91, 93]. A protocol is "TCP-friendly" if the steady-state throughput achieved by this protocol is not greater than the rate of a TCP flow experiencing comparable network conditions. Since then the term "TCP-friendliness" has been used in numerous RFCs ([20, 92, 46, 48, 41], etc.). As already stated, the TCP congestion control algorithm imposes a share of the bottleneck bandwidth to be approximately proportionally fair [121], and therefore, since most flow uses TCP as the transport protocol [96], a roughly proportionally fair share of the capacity is achieved in the Internet.

On the other hand, the TCP protocol (or a protocol with a TCP conforming behavior — TCP-friendly protocol) does not achieve a desirable behavior (meaning: a high throughput, a high link utilization etc.) in certain environments, e.g., links with a high bandwidth-delay product, wireless environments etc. Therefore mechanisms for by-passing these drawbacks are already present in the Internet, for example, opening multiple parallel connections to improve link saturation and obtain a higher throughput can be encountered (e.g., GridFTP [7], peer-to-peer applications); and further, in some operating systems more aggressive TCP variances are to be found. BIC [128] is set as default in the Linux kernel since June 2004, starting with the kernel version 2.6.7; then it was replaced by CUBIC [110] in February 2007. Compound TCP [119] is present in Windows Vista, although not set as the default protocol. The approach "one-size-fits-all" seems to not satisfy the requirements of some existing applications and of users, especially in a situation where the access link is the bottleneck. A different treatment of certain kinds of applications is desirable from the users' point of view (e.g., live video streaming vs. peer-to-peer downloads, or the user is simply more eager to receive file A than file B). Therefore IETF has already started a course of activities that is not TCP-friendliness confirming, like Low Extra Delay Background Transport — LEDBAT (standardizing transport layer protocols for a less important background traffic, like peer-to-peer traffic), multipath TCP — MPTCP (TCP for transfers over a number of parallel paths), Pre-congestion notification — PCN and Pseudowire Congestion Control Framework. The IRTF, Internet Congestion Control Research Group (ICCRG) expresses a desire of moving away from the "TCP-friendliness" paradigm. It tends towards creating a new design team to work on a different paradigm — a new capacity sharing architecture and new congestion control [1]. Although, at the same time, the guidelines for evaluating new experimental congestion controls including a constraint of being not harmful for the TCP protocol are still valid.

Fairness can be imposed in different ways. Since the TCP protocol is the most widely used protocol in the Internet [96], a rough flow-based fairness, obtained mostly by the TCP congestion

control algorithm implemented in end hosts, is achieved in the current Internet. Nevertheless other ways of enforcing flow-based fairness are possible too. For example, just by using a certain scheduling mechanism in the router, per flow fairness can be provided (e.g., [60], [40] and [83]) or a combination of the two could be used (a certain rate control in end-hosts and router support). Fairness in the Internet also leads to a complex discussion: about granularity of entities fairness is imposed upon (per flow, per connection between two hosts, per user etc.), about matrices to be used (like rate-base fairness with a rate given in bits per second or packets per second, cost fairness etc.), about fairness for unicast and multicast traffic etc. The reader is referred to [49] for a summary of forms of fairness present throughout RFCs as well as in the literature. Recently there have been some proposals that suggest moving completely away from pure flow-based fairness. For example, a fair rate allocation not between flow, but economical entities is suggested in [23].

Briscoe [23] offers a fresh look at fairness — cost fairness. It is an orthogonal view of fairness in the Internet and similar discussions are to be found in [90], [115] and in the already mentioned work by Kelly et al. [69, 70]. The author of [23] argues that end-to-end flows from different types of applications should not be serviced with the same rates even when the network conditions are exactly the same. This is because the benefit to a user is not necessarily a linear function of the rate at which his/her application sends or receives (e.g., in the case of SMS vs. video). According to [23], users should be made accountable for causing congestion, and this would give them an incentive to use a protocol like MulTFRC which has a weight parameter. Re-feedback [24] is a framework within which such a protocol could be used.

Chapter 3

Modeling parallel TCP flows

One of the main achievements of this thesis is the modeling of parallel TCP flows. As already seen in the previous chapter the TCP protocol has been comprehensively studied over the years, but there are still some aspects of TCP that are not captured, such as the ones presented here. The main objective of this work is to obtain a practical model of multiple parallel TCP flows traversing the same end-to-end Internet path. The outcome should be a simple but accurate equation that calculates the throughput of TCP flows assuming the flows share the network with an unknown cross-traffic. It should be noted that end-to-end flows between the same two hosts can sometimes traverse different paths, e.g., when traffic engineering with Equal-Cost Multi-Path (ECMP) is applied; such situations are not captured by this model.

As the literature study showed, all existing models of parallel TCP flows assume a certain knowledge about network topology and/or they are rather too complex. Therefore extending one of the models considering just a single TCP flow is a promising approach. Among many proposed models of a single TCP flow the model presented in [103] certainly stands out. Because of an achieved balance between accuracy and simplicity, the equation for the steady-state throughput of a single TCP flow, proposed in [103], is the most used TCP throughput equation. Hence the mathematical approach used in the derivation by Padhye et al. serves as the foundation for the work presented here. This equation [103] uses the state of the network, characterized by the round-trip time (RTT) and a loss event rate (p) as well as the number of packets acknowledged per ACK (b), as the known input parameters. The loss event rate is the frequency at which loss indications occur (a packet is a loss indication if it is the first packet lost in a loss event). The outcome of this equation is the throughput of a single flow estimated in packets per second.

The equation for calculating the throughput of parallel TCP flows, presented here, considers the same characteristics of the underlying network (i.e. RTT, the loss event rate and b) to be known, but dealing with multiple flows introduces certain new aspects. The parallel flows can be observed as individual flows or as an aggregate, and therefore the loss event rate can be

measured in two ways: as the loss event rate on a per-flow basis and as the loss event rate of the aggregate flow. A short discussion of this subtle disparity will be provided in Section 3.1. Consequently, two slightly different models are developed: the model using a per-flow loss event measure (Section 3.2) and the model using a loss event measure of the cumulative flow (Section 3.3). Both models are extended to incorporate the possibility of time-out loss event indications. Further, the validations and real-life measurements provided in Sections 3.2.4, 3.3.4 and 3.3.5 show the accuracy of both models.

3.1 Measuring loss of parallel flows

For estimating the throughput of a TCP flow, packets that cause a reduction of the window size of the flow are of special interest; in other words, packets which produce loss events via triple-duplicate acknowledgements or time-outs. TCP reduces its window just once in a round-trip time, and therefore, not all lost packets are loss event indicators; if a packet is a loss event indicator consecutive losses which are detected less than a round-trip time after the loss event indicator can not be new loss event indicators. As input into the model the loss event rate is of special interest.

n parallel TCP flows can be observed from two perspectives: observing each flow separately or observing the set of flows as an aggregate. Hence there are two ways of measuring loss events experienced by the set of flows:

- **per-flow loss event rate measure** — how often loss event indicators belonging to any of the n flows are measured. Each flow is observed separately and loss events of each flow are counted. Since multiple lost packets in a round-trip time cause the congestion window of a flow to be reduced just once, after a loss experienced by a flow, consecutive losses of that flow within the same round-trip time can not start a new loss event and are neglected for the loss event rate measure.
- **loss event rate measure of the cumulative flow** — all n flows are treated as a single flow and the loss event rate of this cumulative flow is estimated. As with the per-flow loss event rate measure, lost packets within the same round-trip time as a packet loss triggering a window reduction are disregarded. Therefore in a loss event of the cumulative flow more than one individual flow can reduce its window.

  An example of four flows is depicted in Figure 3.1. We assume that each of these four flows experiences a loss event in a time period which has the duration of the round-trip time (flow 1 loses two packet but just the first one is counted as a loss event). The first model counts three loss events. The cumulative flow, built from these four flows, also loses four packets but just the first lost packet is its loss event indicator. The second model counts just one loss event — yet in that loss event of the cumulative flow, three individual flows experience loss events.

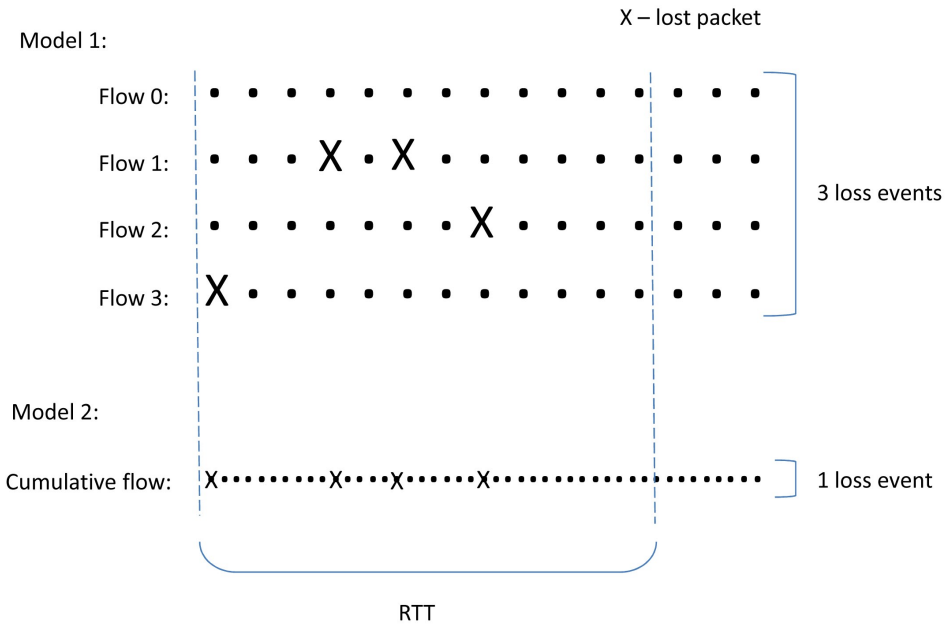


Figure 3.1: The per-flow loss measure vs. the cumulative flow loss measure

For a high loss rate and a large number of considered flows the two loss measuring methods will therefore differ in such a way that the per-flow loss measurement would have a higher value.

These two measuring methods differ just slightly, but, by eliminating the need for a distinction between flows in the second case, a measurement complexity reduction is gained by using just on instead of a number of counters. Therefore the second alternative is a more practical method.

Accordingly, two models are derived as an outcome of this thesis. From a mathematical perspective the use of the per-flow loss measure leads to a simpler extension of the equation from [103]; a detailed derivation is given in Section 3.2. Further, a refinement of this derivation to facilitate easier loss measurements is achieved in Section 3.3. Adding a measurement of the rate at which packets are lost besides the estimation of the loss event rate produces just a minor measurement overhead and at the same time increases the accuracy of the second equation. It enables estimating the number of flows experiencing loss event indications in a loss event of the cumulative flow, and hence yields a more accurate calculation of the time-out probability.

Another advantage of the model using the loss measure of the cumulative flow is shown in Chapter 4, where it is used as a building block of the MulTFRC protocol. A flow using the MulTFRC protocol behaves like n TCP flows; since we actually have one, not n flows, it is not possible to distinguish which of these n "virtual" flows a packet belongs to. Therefore it is not possible to measure the loss event rate of individual flows, but only of the cumulative flow and for that, the second equation will be used.

3.2 The model — per-flow loss measure

In order to derive an equation for the throughput of several parallel TCP flows we extend the model presented in [103].

Consider n parallel TCP flows $f_1, ..., f_n$ starting at time $t = 0$. As in [103] we model TCP's congestion avoidance phase in terms of "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; see also Figure 3.2). We note that, while this assumption may seem a bit far-fetched, it is a necessity for keeping the model simple. As we will see later, this does not prevent the model from capturing the behavior of multiple TCP's accurately enough to make it a useful tool with various practical applications.

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. 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 time-outs. 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. In Section 3.2.2 we will extend this equation to incorporate time-out loss indications.

3.2.1 Model considering only the congestion avoidance phase

We assume that all flows share the same path and have the same average RTT. They are mixed (i.e. in a round packet 1 belongs to f_1, packet 2 to f_2, and so on). This is shown in Figure 3.2. Whenever a flow f experiences a TD loss indication, it halves its window size W_f. For n parallel flows we define a TD-period (TDP) as a period between two consecutive TD loss indications in any flow. We assume that in each TDP just one flow experiences loss. p is defined as the probability that a packet is the first packet lost in a loss event of any flow (p is the probability of a loss event of any flow).

Note that we do not take the same assumption as in [103] (if a packet is lost, all consecutive packets belonging to the same window are lost too). This is because:

- Other than the authors of [103], we model multiple flows, which are more aggressive than a single flow, giving their sum (the cumulative flow) a better chance to attain a large

window size. This would amplify errors introduced by wrongly assuming a large cluster of packets to be lost.

- We believe that this decision is well justified: from [25] 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").
- Finally, since we assume packets from flows to be mixed in a round-robin fashion, the probability for a cluster of consecutively dropped packets to affect a single flow decreases as the number of flows increases.

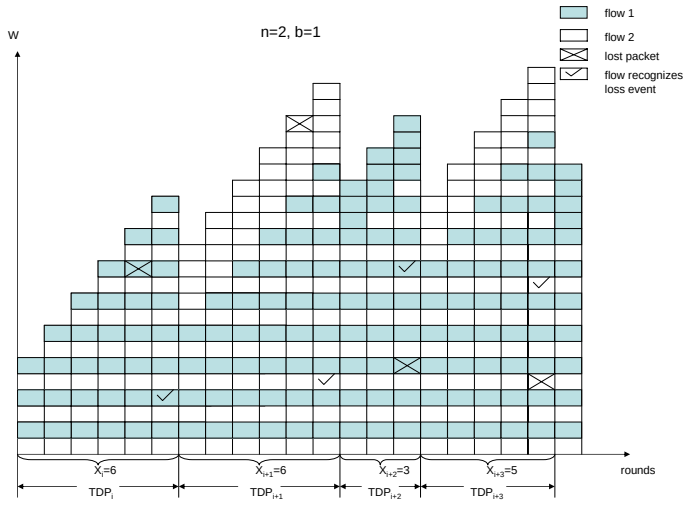


Figure 3.2: Triple Duplicate Periods ($TDPs$)

For a period TDP_i let Y_i be the number of packets sent in that period, A_i be the duration of that period and X_i be the number of rounds in that period. It can be shown ([103]) that

$$B = \frac{E[Y]}{E[A]} \tag{3.1}$$

where $E[Y]$ and $E[A]$ are the average number of packet sent in a TD-period and the average duration of a TD-period.

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

At the end of a TD-period of the cumulative flow just one flow experiences loss, so a flow will not experience loss events at the end of each TD-period. If we assume that loss occurs

identically distributed over all flows, the probability that a flow experiences loss at the end of a TD-period is $1/n$. The probability that the time between two loss events of a flow (A_f) is k TD-periods ($k = 1, 2, ...$) is equal to the probability that the flow did not lose a packet in $(k-1)$ consecutive TD-periods and in the k-th TD-period it loses a packet:

$$P[loss\ in\ the\ k-th\ TDP] = \frac{1}{n}(1-\frac{1}{n})^{k-1}. \tag{3.2}$$

The mean of A_f is:

$$\begin{aligned} E[A_f] &= \sum_{k=1}^{\infty}\left(\frac{1}{n}\left(1-\frac{1}{n}\right)^{k-1} kE[A]\right) \\ &= \left(nE[A]\right). \end{aligned} \tag{3.3}$$

The duration of the i-th TD-period is: $A_i = \sum_{j=1}^{X_i} d_{ij}$, where d_{ij} is the duration of the j-th round in the i-th TD-period. As in [103], if we consider the duration of each round to be a random variable independent of the window size and also the round number with the average value RTT and $E[X]$ to be the average number of rounds in a TD-period, we have:

$$E[A] = E[X]RTT\ . \tag{3.4}$$

For deriving $E[Y]$ we will examine the evolution of the cumulative window (W) on the sender side, shown in Figure 3.3. 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 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

$$E[Y] = E[\gamma]\ . \tag{3.5}$$

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

$$P[\gamma_i = k] = (1-p)^{k-1}p, k = 1, 2, ... \tag{3.6}$$

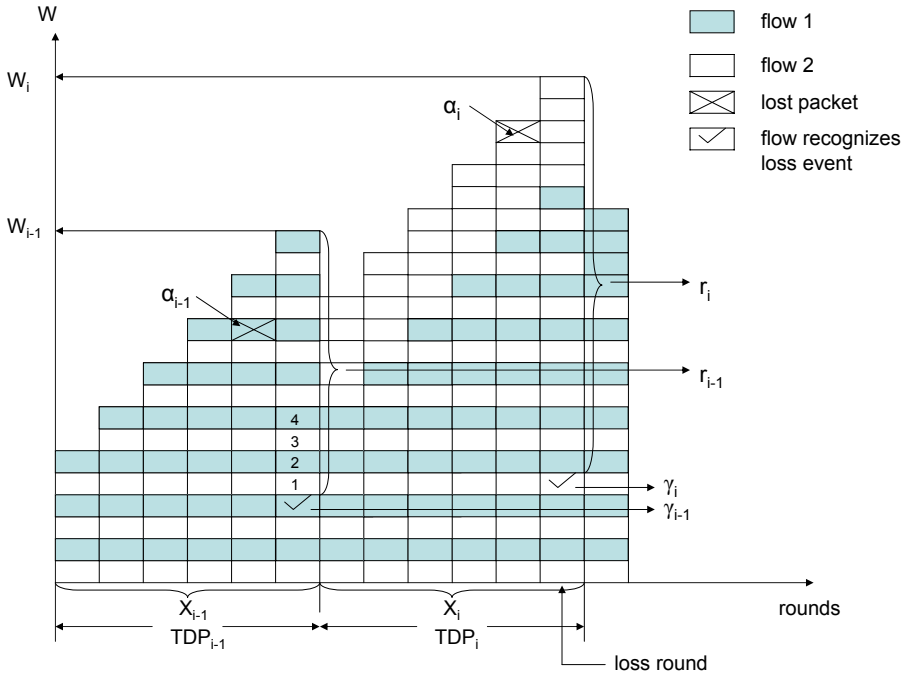


Figure 3.3: A Triple Duplicate Period (TDP)

And the mean value of γ is:

$$E[\gamma] = \sum_{k=1}^{\infty}(1-p)^{k-1}pk = \frac{1}{p} \ . \tag{3.7}$$

Looking at the evolution of the window size W we have to assume that in the loss round of each TD-period one flow experiences loss. For the $(i-1)$th TD-period let this flow be flow m. Flow m does not experience loss in every TD-period, so the TD-periods in which flow m experiences loss are a subset ($\{i_s\}, s = 1, 2...$) of TD-periods of the cumulative flow. For example, in Figure 3.2 the subset of TD-periods for flow 1 is TDP_i, TDP_{i+2} and for flow 2 it is $\{TDP_{i+1}\}$, $\{TDP_{i+3}\}$.

If $W_{f_{m_{i_s}}}$ is the window size of flow m at the end of the (i_s)th TD-period, and $X_{f_{m_{i_s}}}$ is the number of rounds from the end of $TDP_{i_{s-1}}$ till the end of TDP_{i_s}, during $X_{f_{m_{i_s}}}$ rounds the window size of flow m increases by $\frac{X_{f_{m_{i_s}}}}{b}$ (as in [103]) and we have:

$$W_{f_{m_{i_s}}} = \frac{W_{f_{m_{i_{s-1}}}}}{2} + \frac{X_{f_{m_{i_s}}}}{b} \ . \tag{3.8}$$

The number of packets sent in a TD-period is the number of packets sent between two loss events. For the i-th 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) and packets sent in next X_i rounds till the (γ_i)th packet. If flow m is the flow that experiences loss in the $(i-1)$th TD-period and $W_{f_{m_{i-1}}}$ is the window of that flow at the end of the $(i-1)$th TD-period, the window size of the cumulative flow at the beginning of the i-th TD-period is $W_{i-1} - \frac{W_{f_{m_{i-1}}}}{2} + (n-1)$ (flow m reduces its window and the remaining $(n-1)$ flows increase their window size by 1). Every b rounds the window size of the cumulative flow is increased by n and we have:

$$Y_i = r_{i-1} + \sum_{k=0}^{\lfloor X_i/b \rfloor - 1} \left(W_{i-1} - \frac{W_{f_{m_{i-1}}}}{2} + (n-1) + nk \right) b - r_i \tag{3.9}$$

where r_i is the number of packets sent in the loss round of TDP_i after the loss event is recognized.

Assuming that $\{X_{f_{m_{i_j}}}\}$ and $\{W_{f_{m_{i_j}}}\}$ are mutually independent sequences of i.i.d. random variables (as in [103]), from (3.8) we have:

$$E[W_{f_m}] = \frac{2}{b} E[X_{f_m}] \tag{3.10}$$

where $E[W_{f_m}]$ is the average window size of flow m and $E[X_{f_m}]$ the average number of rounds between two loss events of flow m. Taking into account the assumption that a loss occurs identically distributed over all flows we can say that $E[X_{f_m}]$ and $E[W_{f_m}]$ are equal for all flows and from now on we denote them by $E[X_f]$ and $E[W_f]$.

From (3.3) and (3.4) we have:

$$E[A_f] = nE[X]RTT \tag{3.11}$$

and the average number of rounds between two loss events of a flow ($E[X_f]$) is

$$E[X_f] = nE[X] \ . \tag{3.12}$$

Assuming that at the end of each TD-period the window size of a flow experiencing loss is $E[W_f]$ and the window sizes of flows experiencing loss in previous loss events are: $\frac{E[W_f]}{2} + \frac{E[X]}{b}$, $\frac{E[W_f]}{2} + \frac{2E[X]}{b}$ and so on, the mean value of the cumulative window size of all flows is:

$$E[W] = E[W_f] + \sum_{k=1}^{n-1} \left(\frac{E[W_f]}{2} + \frac{kE[X]}{b} \right) \ . \tag{3.13}$$

Considering (3.13), (3.10) and (3.12) we have:

$$E[W] = \frac{nE[X]}{2b} + \frac{3E[X]n^2}{2b} \ . \tag{3.14}$$

From (3.9), 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:

$$E[Y] = \left(E[W] - \frac{E[W_f]}{2} + (n-1)\right)E[X] + \frac{nE[X]^2}{2b} - \frac{nE[X]}{2} \tag{3.15}$$

and considering (3.5), (3.7), (3.10), (3.12) and (3.14):

$$\frac{1}{p} = \frac{3n^2E[X]^2}{2b} + \frac{nE[X]}{2} - E[X] \ . \tag{3.16}$$

Solving this equation for $E[X]$ we get:

$$E[X] = \frac{2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p}}{6n^2p} \ , \tag{3.17}$$

and from (3.14) we have:

$$E[W] = \frac{2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p}}{4bp} + \frac{2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p}}{12bnp} \ . \tag{3.18}$$

Finally, from (3.1), (3.5), (3.7), (3.4) and (3.17) we have:

$$B = 1/RTT \cdot \frac{6n^2}{(2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p})} \ . \tag{3.19}$$

3.2.2 Model with time-outs

We now extend the equation to include time-outs (TO). A loss event experienced by a flow can be a triple-duplicate loss indication or a time-out 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 i-th TD-period and with R_{TO_i} the number of packets sent by flows that are in the slow start phase in the i-th TD-period. The long term steady-state throughput B_{ext} is:

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

where $E[B_{TDP}]$ is the throughput of flows that are not in the slow start phase. And, considering 3.4, we have:

$$B_{ext} = E[B_{TDP}] + \frac{E[R_{TO}]}{E[X]RTT} \ . \tag{3.20}$$

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

$$E[B_{TDP}] = B\frac{n - E[nTO]}{n} \tag{3.21}$$

where B is defined in (3.19) and $E[nTO]$ is the average number of flows that are in the slow start phase. The second part of (3.20) ($\frac{E[R_{TO}]}{E[X]RTT}$) is the throughput of flows that are in slow start and it equals: $E[nTO]\frac{E[R]}{E[Z^{TO}]}$, where $E[R]$ and $E[Z^{TO}]$, following the notation from [103], are the average number of packets sent by one flow $E[R]$ during a time-out sequence of average duration $E[Z^{TO}]$.

$E[R]$ and $E[Z^{TO}]$ are calculated in the same way as in [103]. A flow experiencing a time-out loss indication will send a single packet after time T and if that packet or its ACK is dropped (the probability of this event is p) the sender waits for $2T$ time and sends another packet. Therefore, the probability that a flow experiencing a time-out loss indication is going to send k ($k > 1$) packets is:

$$P[R = k] = p^{k-1}(1-p) \tag{3.22}$$

and the average is calculated by:

$$E[R] = \sum_{k=1}^{\infty} kp^{k-1}(1-p) \ . \tag{3.23}$$

A flow that is going to experience a time-out loss indication, waits for time T before retransmitting a packet. If this packet is successfully transmitted the duration of the time-out was T. Further, if the first packet is dropped, the sender waits for $2T$ before retransmitting it (the total time-out duration is now $3T$) and this is repeated until the wait period reaches a duration of $64T$ (which is 2^6T). The probability that $k-1$ ($\{k \leq 7\}$; 7 consecutive time-outs are necessary to reach $64T$) consecutive packets are dropped and the k-th one is successfully transmitted is $P[loss\_packet = k] = p^{k-1}(1-p)$, and in that case the duration of the time-out period is $\sum_{i=1}^{k} 2^{i-1}T = (2^k - 1)T$. In the case $k > 7$ the duration is $127T + (k-7)64T$ and the probability of such a event is calculated in the same way as when $k < 7$. Therefore the average duration of a time-out period is:

$$E[Z^{TO}] = \sum_{k=1}^{7} p^{k-1}(1-p)(2^k - 1)jT + \sum_{k=8}^{\infty} p^{k-1}(1-p)(127T + (k-7)64T) \ . \tag{3.24}$$

From (3.23) and (3.24), we get:

$$E[R] = \frac{1}{(1-p)}, \tag{3.25}$$

$$E[Z^{TO}] = T * \frac{f(p)}{1-p} \tag{3.26}$$

where

$$f(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6$$

and T denotes the initial period of time (in a TO phase) after which the sender retransmits unacknowledged packets; as in [103], $f(p)$ can be approximated with $1 + 32p^2$.

The main challenge when attempting to calculate the throughput of parallel TCP flows including time-outs is to find the expression for $E[nTO]$ — the number of flows that are in the slow start phase during one TD-period.

Estimating the number of flows in the slow start phase

Before deriving the final equation a short discourse on the calculation of $E[nTO]$ is given. Some of the most relevant methods that have been considered in the course of this work are discussed.

The estimation of $E[nTO]$ is strongly coupled with the calculation of the probability that a flow experiences a time-out loss indication (denoted by P_{TO}), and therefore the time-out probability of a single flow is considered first.

For deriving the time-out probability in [103], a DropTail loss assumption is taken. There it is considered that all packets in a loss round after the first lost packet are also lost, in other words a correlated loss is assumed. Since packet losses observed in the Internet experience rather short bursts [25] (as already stated, 83% of losses are just drops of a single packet), it is tempting to believe that a non-correlated loss assumption would produce better results. In [25] the Bernoulli loss assumption is suggested — loss of a packet is independent of unsuccessful arrival of the other packets and all packets have the same probability to be dropped. Therefore, both the DropTail and Bernoulli loss assumptions are examined here.

We will observe packets of a flow that experiences loss. The window size of that flow in the loss round, from (3.10), (3.12) and (3.17), is:

$$E[W_f] = \frac{2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p}}{3npb} . \tag{3.27}$$

Further, the packet loss probability can be observed from two perspectives: considering all packets of the flow or considering just the packets belonging to the loss round. By observing all packets, the probability that any packet is lost is p. Observing packet losses from this point of view is used in [103] and [25]. For the second possibility, p_{lr} denotes the probability that one of the packets belonging to the loss round is lost. Since at least one of the packets of the flow's window is lost, the probability that a packet of that flow is lost in its loss round is at least $\frac{1}{E[W_f]}$. We take this as p_{lr}.

For deriving the final solution, we analyzed both loss assumptions combined with both packet loss observations possibilities. In the following text, this analysis will be explained in more detail, and a validation experiment is shown in Figure 3.4. The figure depicts a comparison between the models with all considered time-out calculations and the throughput of 5 TCP flows obtained using the ns-2 simulator [4]. For these simulations the same network topology as for generating Figure 3.6 (a) is used (for more details the reader is referred to Section 3.2.4).

Assuming that any packet can be lost with the same probability p, the authors of [103] and [25] derive the probability of a time-out loss indication by calculating the probability that in a round, less than three packets are successfully transmitted under a constraint that in the round a loss occurs or that in the second last round, more than two packets are acked but still less than

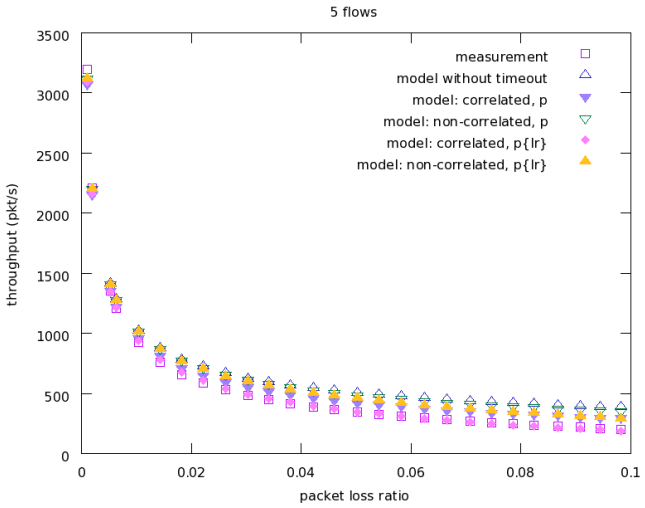


Figure 3.4: Comparison of variants for the time-out probability calculation; correlated and non-correlated loss assumption combined with two ways of measuring the packet loss probability: the packet loss probability of any packet and the packet loss probability in a loss round

three successfully received and acked in the consecutive round. A similar calculation extended for the case of multiple flows is also considered here. The P_{TO} is calculated using the derivation from [103] and [25] for the correlated and the non-correlated loss assumption respectively. The packet loss probability is p and the window size of the flow is $E[W_f]$ (equation 3.27). The result is the probability that a single flow experiences a time-out loss indication. Hence the average number of flows in the slow start phase is

$$E[nTO] = P_{TO} * n \ .$$

In Figure 3.4 the lines designated by "model: correlated, probability p" and "model: non-correlated, p" show the model with these two time-out calculation variants.

Further, we examine the second two possibilities for calculating the time-out probability by observing packets belonging just to a round in which the flow experiences a loss event. As in the previous analysis, both the non-correlated loss assumption and the correlated loss assumption are considered.

The Bernoulli loss assumption:

A flow experiencing loss in a loss round of the cumulative flow is observed. Let $B(w,k)$ be the probability that k packets of w are successfully transmitted and the other $w-k$ are lost. According to the non-correlated loss mode and considering that packet losses are independent and each packet is dropped with the same probability p_{lr}, $B(w,k)$ is:

$$B(w,k) = \binom{w}{k} p_{lr}^{w-k}(1-p_{lr})^k \ . \tag{3.28}$$

The flow will experience a time-out loss indication if in the loss round less than three packets are successfully transmitted ($\sum_{k=0}^{2} B(w,k)$) or in the second last round more than two packets are acked ($\sum_{k=3}^{w-1} B(w,k)$), but in the successive round less than two packets are successfully delivered ($\sum_{m=0}^{2} B(k,m)$). Hence $P_{TO}(w)$ — the probability the flow experiences a time-out depending on its window size (w) — is:

$$P_{TO}(w) = \begin{cases} 1 & if \quad w \leq 3 \\ \sum_{k=0}^{2} B(w,k) + \sum_{k=3}^{w-1} B(w,k) \sum_{m=0}^{2} B(k,m) & if \quad w \geq 4 \end{cases}$$

Further, similar to [25], we assume that the window size of the flow is uniformly distributed on the discrete interval $[0, w_{max}]$. Therefore the probability of the flow suffering a time-out is:

$$P_{TO} = \sum_{w=2}^{w_{max}} \hat{P}_{TO}(w) P[W=w] \ . \tag{3.29}$$

In a loss event just one flow experiences loss and this loss leads to a time-out loss indication with the probability P_{TO}. $E[nTO]$ should include not just a flow that experiences a time-out in the current loss round, but also flows that experienced time-outs in one of the previous TD-periods and that are still in the slow start phase. Then we have: $E[nTO] = P_{TO} \frac{E[Z^{TO}]}{E[X]*RTT}$. We consider that the maximal window size of a flow experiencing loss (w_{max}) is given by equation (3.27).

The DropTail loss assumption:

The DropTail loss assumption corresponds to the one presented in [103]: if a packet is dropped all consecutive packets belonging to the same round are dropped too. The probability that k among w packets belonging to a flow experiencing loss in a round are successfully transmitted and acked is:

$$A(w,k) \;=\; (1-p_{lr})^k p_{lr} \quad 1 \le k < w$$

and $A(w,w) = 0$ because at least one packet of the flow must be lost The probability that a flow experiences a time-out in a loss round is equal to the probability that less than three packets are successfully received and acked in the loss round ($\sum_{k=0}^{2} A(w,k)$), or that in the second last round more than three packets are successfully received, but in the last round less than three are successfully delivered ($\sum_{k=3}^{w-1} A(w,k) \sum_{m=0}^{2} A(k,m)$). Then, the probability that a loss in a window of size w is a TO is:

$$\hat{Q}_{TO}(w) = \qquad (3.30)$$
$$\begin{cases} 1 & w \le 3 \\ \sum_{k=0}^{2} A(w,k) + \sum_{k=3}^{w-1} A(w,k) \sum_{m=0}^{2} A(k,m) & otherwise \end{cases}$$

and we obtain:

$$\hat{Q}_{TO}(w) =$$
$$min(1, (1-(1-p_{lr})^3)(1+(1-p_{lr})^3(1-(1-p_{lr})^{w-3})) \; . \qquad (3.31)$$

As in the previous case, the average number of flows in the slow start phase is:

$$E[nTO] = \hat{Q}_{TO}(w) \frac{E[Z^{TO}]}{E[X] * RTT}, \qquad (3.32)$$

and the window size w is equal to $E[W_f]$ (3.27).

Figure 3.4 also depicts the models with the time-out probability calculations using the last two derivations. The line named "model: correlated; p_{lr}" demonstrates the accuracy of the model using the Bernoulli loss assumption and the last line — "model: non-correlated, p_{lr}" shows the results of the model including the slow start phase calculated according to the DropTail loss assumption.

Comparing the above described variants of the time-out probability calculation, we came to the conclusion that the DropTail loss assumption is accurate in a broad range of the network conditions. This may seem to be at odds with our previous assumption of loss events being independent and randomly distributed; however, this result is in accordance with the general assumptions that are central to the design of TCP itself, that is, we consider loss events that cause time-outs to be rare special cases where packets are consecutively dropped (TCP exhibits a Go-Back-N behavior in Slow Start because of this assumption).

The TCP Steady-State Throughput Equation for Parallel TCP Flows

The final model including time-outs is obtained by using the DropTail loss assumption and observing only packets of a loss round (equation 3.31). From (3.20), (3.21) and (3.32), considering that a flow experiencing loss has the window size ($E[W_f]$) given in (3.27), we have:

$$B_{ext} = B\frac{n - \hat{Q}_{TO}(E[W_f])\frac{E[Z^{TO}]}{E[X]*RTT}}{n} + \hat{Q}_{TO}(E[W_f])\frac{E[Z^{TO}]}{E[X]*RTT}\frac{E[R]}{E[Z^{TO}]} \tag{3.33}$$

where B, $\hat{Q}_{TO}(w)$, $E[Z^{TO}]$, $E[R]$, $E[X]$ are given in (3.19), (3.31), (3.26), (3.25), (3.17) respectively.

Finally, using $\hat{Q}_{TO}(w)$ and $E[W_f]$ defined in (3.31) and (3.27), we obtain the long term steady-state throughput of n TCP flows as a function of RTT, p and b:

$$B_{ext} = \frac{6n^2(1+\hat{Q}_{TO}(E[W_f])\frac{p}{1-p})}{(2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p})RTT} - \frac{36n^3p\frac{T(1+p+2p^2+4p^3+8p^4+16p^5+32p^6)}{(1-p)}\hat{Q}_{TO}(E[W_f])}{(2pb - npb + \sqrt{n^2p^2b^2 - 4np^2b^2 + 4p^2b^2 + 24bn^2p})^2RTT^2}. \tag{3.34}$$

3.2.3 Algorithm

The equation derived in the previous section is now displayed in the form of an algorithm. As input parameters the algorithm requires:

- n — the number of flows
- p — the loss event rate measured on a per-flow basis
- b — the number of packets acknowledged by a received ACK
- RTT — round-trip time
- T — the initial period of time (in a TO phase) after which the sender retransmits unacknowledged packets

Variables a, x, w, z, $q1$ and q are temporary floating point variables. The throughput is calculated in packets per second.

It should be noted that we are not suggesting a complete implementation of our model here, but merely presenting the equations in the form of an algorithm. Some issues like, for instance, the best way to obtain the RTT measurement, are still left up to the programmer who wants to make use of the algorithm.

Algorithm 1 The model using the per-flow loss measure: The throughput of n parallel flows [pkt/s]

$a = sqrt(p * b * (24 * n * n + p * b(n * n - 4 * n + 4)))$
$x = (p * b * (2 - n) + a)/(6 * n * n * p)$
$w = 2 * n * x/b$
$z = T * (1 + 32 * p * p)/(1 - p)$
$q1 = (1 - (1 - pow(1 - 1/w, 3))) * (1 + pow(1 - 1/w, 3) * (1 - pow(1 - 1/w, w - 3)))$
if $q1 > 1$ **then**
  $q1 = 1$
end if
if $q1 * z/(x * RTT) \geq n$ **then**
  $q = n$
else
  $q = q1 * z/(x * RTT)$
end if
return $(1 - q/n)/(p * x * RTT) + q/(z * (1 - p))$

3.2.4 Validation

We validated the accuracy of the TCP model presented in the previous section using the ns-2 simulator [4], version 2.31. With simulations we showed that the presented equation works well in a broad range of conditions. Real background traffic can produce different distributions of loss events. These events include isolated packet losses, burst losses as well as variations of the specific length of the burst. All these loss distributions influence the throughput of an aggregate of TCP flows in a different way. To validate our equation we chose a non-correlated loss model (each packet is lost with the same probability), a loss model with burst losses as well as specific loss patterns that are produced by a bottleneck link with a DropTail queue and a RED queue.

For our ns-2 simulations we used the "dumbbell" topology which is commonly used for studying the behavior of transport protocols. The dumbbell configuration represents a set of data flows sharing the same link. It is depicted in Figure 3.5. In the figure, "S" represents a sender and "R" a receiver. Link "X-Y" is a link shared by all connections. The topology parameters (the link bandwidth and delay) were set in such a way that link "X-Y" is always the bottleneck for all connections. In certain simulations a probability of packet loss was introduced on the

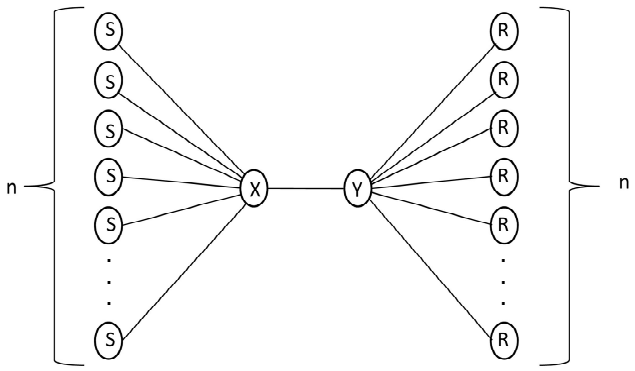


Figure 3.5: Dumbbell topology

bottleneck link. We varied parameters, like the loss percentage and the number of flows to validate our model in different network conditions. In simulations with the loss produced only by flows sharing the common bottleneck, the bottleneck capacity was varied.

A number of parallel TCP flows were run, and the throughput and the loss event rate experienced by these flows was measured (the loss event rate was measured on a per-flow basis). The packet size for all flows was 1000 bytes, and additionally 40 bytes are used for the TCP/IP header (the same value was used in [50]; we chose this value for consistency and easier comparison of MulTFRC and TFRC). The throughput was measured by looking at the number of packets leaving the sender, irrespective of their sequence number and in all figures the average throughput in individual simulation runs are shown. We also measured the average loss event rate by tracing TCP parameters (the ns-2 simulator gives a possibility of tracing fast recovery and time-out loss events). We use the measured average loss event rate and the average round-trip time as input parameters to calculate the throughput using our equation with and without the time-out phase; these are the two other values shown in the figures.

Simulations with random loss

In this scenario, the access links had a bandwidth of 10 Gbit/s and a delay of 1 ms, whereas the bandwidth and delay of the shared link were 1 Gbit/s and 30 ms, respectively. This way, the shared link was always the bottleneck, although, since the bottleneck capacity was very high, the packet loss was only caused by the loss model and not by queues overflows. Each simulation was run for 460 seconds. The first 15 seconds of all simulations were not used for throughput and loss event rate measurements because we wanted to eliminate the influence of the initial slow start phase on a throughput measurement.

We added random loss to the shared link and each packet had the same probability to be dropped. Additionally, we avoided phase effects by using a RED queue. Although this did

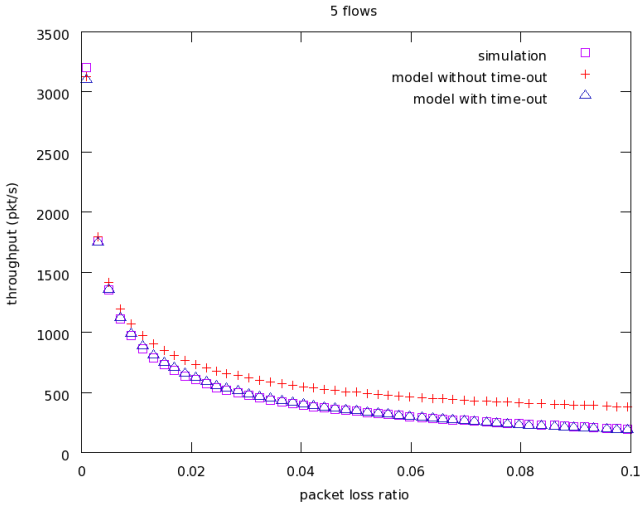
not have an effect on the outcome of the simulations, for the sake of completeness, we also state that the RED queuing mechanism was also used on the access links. According to the recommendation in [3], we allowed the algorithm implemented in ns-2 to automatically configure the RED parameters ($q\_weight = -1$, $thresh = 0$, and $maxthresh = 0$ were set for automatic configuration and the "gentle" mode was used). To verify the equation with a combination of random loss and correlated loss which can be produced by a simple FIFO queue, we used a simulation setup in which the above loss model is combined with a DropTail queuing on the shared link.

The amount of loss that is generated by the loss model on the shared link is the parameter that we varied. To validate the equation with different delay, the bottleneck link delay was also varied (30ms and 100ms).

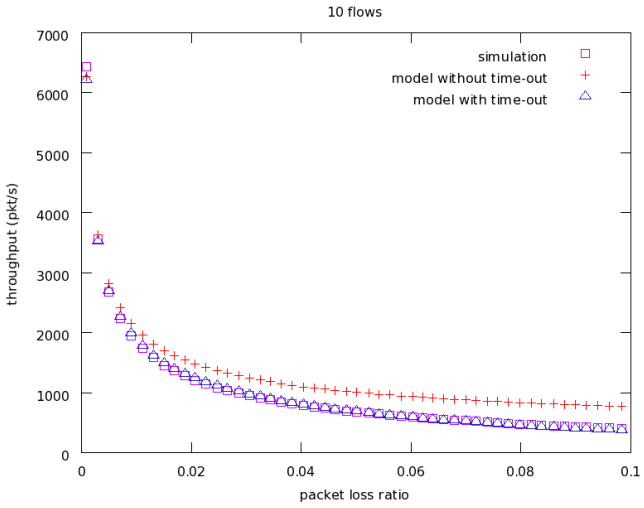
Figures 3.6 and 3.7 show results obtained using the RED queue mechanism on the shared link. In Figure 3.6, it can be seen that that the equation matches the behavior of multiple TCP's closely with a broad range of loss event rates, from a very low loss level to a high one of 10% (even unrealistic in today's Internet). Figure 3.7 shows that the same good behavior scales with the number of flows. We tested it with up to 100 flows. All figures show the packet loss ratio measured in simulations.

For all simulation with 30 ms delay on the shared link, the difference between the measured throughput of the observed flows and its estimate obtained using our equation does not exceed 10%; the error is less than 5% for a loss rate lower than 8%. At around the 5% loss rate the equation switches from overestimating the throughput for a lower loss rate to underestimating it for larger values. This change is probably caused by our suboptimal time-out probability calculation. To confirm our suspicion, we observed Figure 3.8, where the results of simulations with the longer shared link delay (the delay of 100ms) are shown. In the range from 0% till 2%, the measured throughput and the results of our equation differ for the same small factor as in the case of the shorter link delay. With higher loss rates and the longer delay our equation always overestimates. The difference between RTT and RTO in the simulations with the longer delay become smaller (for the 30ms shared link $RTT/RTO = 0.32$ and for the delay of 100ms on the shared link $RTT/RTO = 0.84$) and this difference influences the time-out probability calculation (the reader is referred to equation [3.32]). Therefore, for the longer shared link delay the time-out probability calculation gives a smaller value. This fact confirms our suspicion that this mismatching for a higher loss rate is caused by an error introduced by a less than ideal time-out probability calculation. As already mentioned, some more information would be necessary for a more precise calculation of the time-out probability. Nevertheless Figure 3.8 shows that the equation also captures the behavior of TCP well when the RTT is longer (around 200ms). In this case the difference between simulations and the equation is slightly higher for larger loss rates.

For the sake of completeness, we also ran the same simulations with the DropTail queue mechanism on the shared link. The queue had a length equal to the bandwidth $\times$ delay product. The results are shown in Figures 3.9, 3.10 and 3.11. As expected the results are completely the same as when the RED queuing mechanism was used. We chose a very high capacity on the shared link so that the loss on the link was only produced by the applied loss model; in other

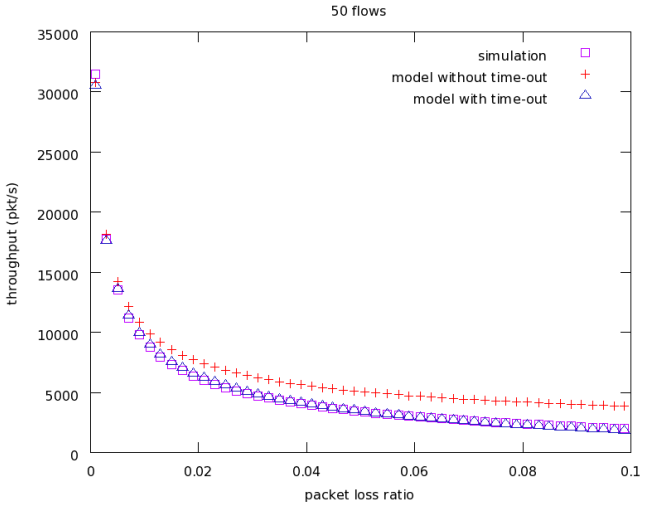


(a)

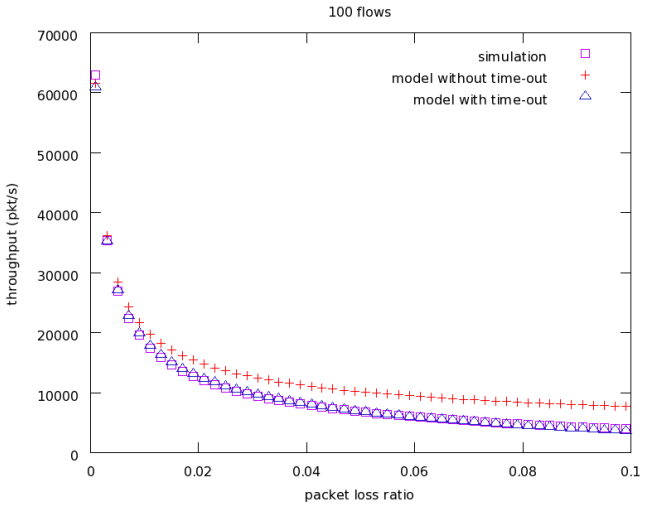


(b)

Figure 3.6: The model with the per-flow loss measure: random loss (each packet is lost with the same probability), RED queue, 30ms delay on the shared link

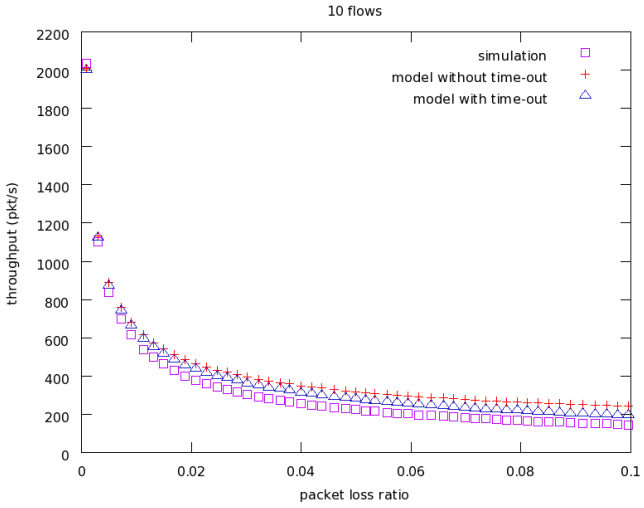


(a)

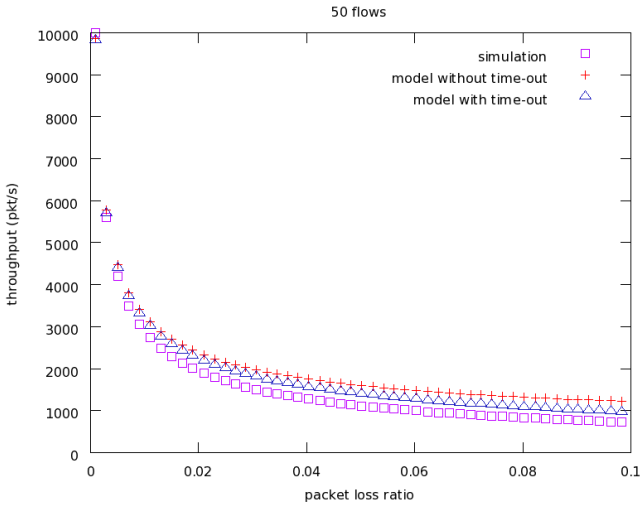


(b)

Figure 3.7: The model with per-flow loss measure: random loss (each packet is lost with the same probability), RED queue, 30ms delay on the shared link

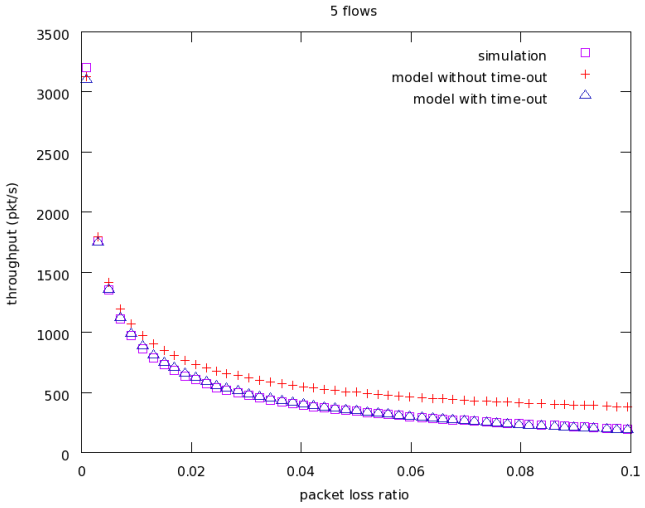


(a)

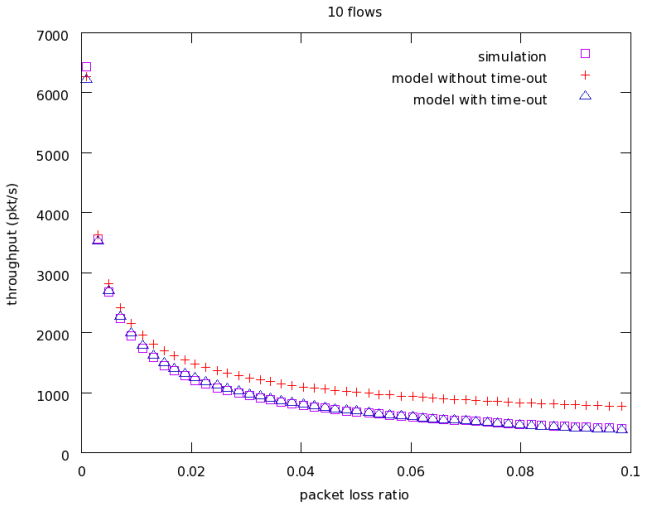


(b)

Figure 3.8: The model with the per-flow loss measure: random loss (each packet is lost with the same probability), RED queue, 100ms delay on the shared link

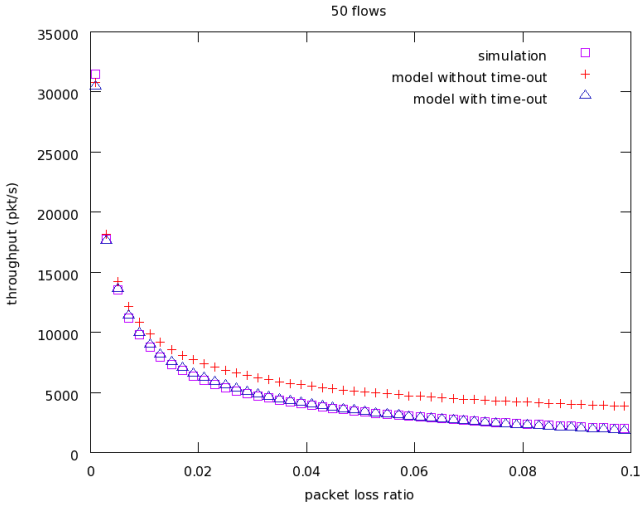


(a)

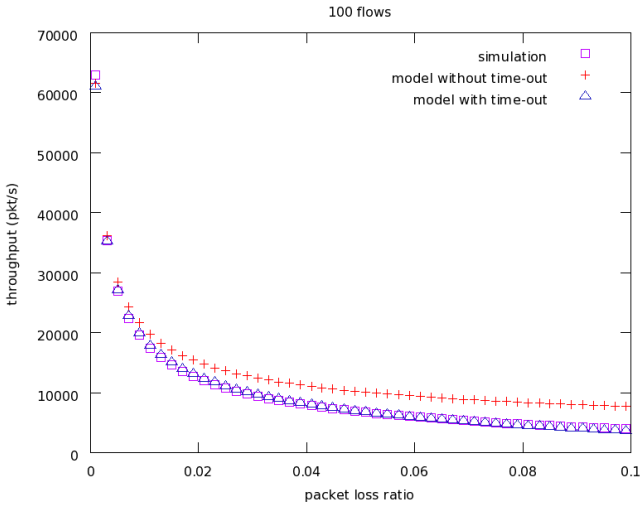


(b)

Figure 3.9: The model with the per-flow loss measure: random loss (each packet is lost with the same probability), DropTail queue, 30ms delay on the shared link

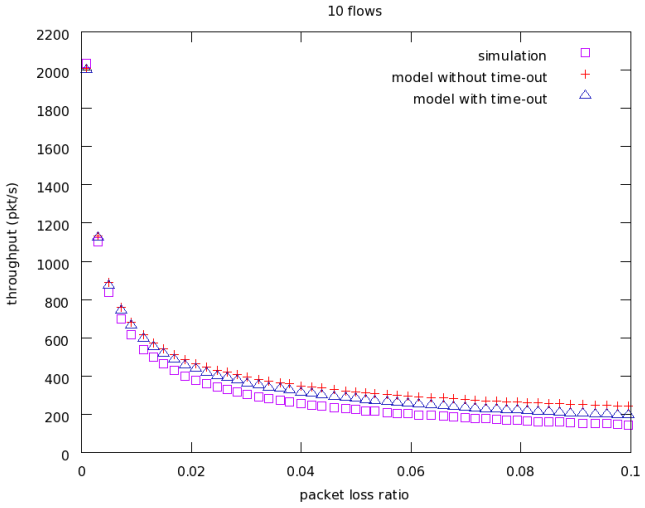


(a)

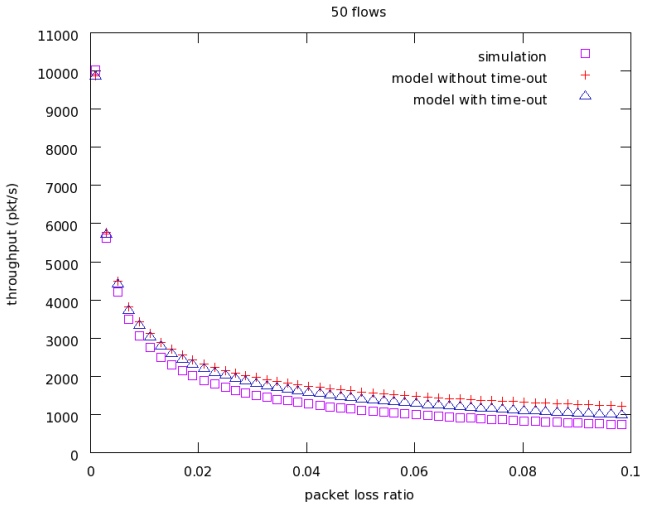


(b)

Figure 3.10: The model with per-flow loss measure: random loss (each packet is lost with the same probability), DropTail queue, 30ms delay on the shared link



(a)



(b)

Figure 3.11: The model with the per-flow loss measure: random loss (each packet is lost with the same probability), DropTail queue, 100ms delay on the shared link

words, the observed flows never exceeded the capacity of the shared link and the round-trip time on the path did not vary during the simulations. Therefore, simulations with different queuing mechanisms used on the shared link show the same results.

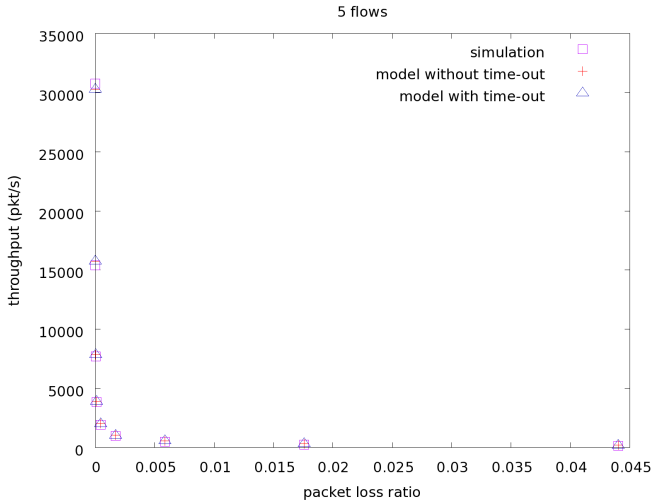
Simulations with loss from a DropTail (FIFO) queue and a RED queue

In the second part of our simulations, we did not use any artificial loss and we let packet loss to be produced just by observed flows alone. Since a DropTail queue and a RED queue have a quite different behavior simulation setups with both queuing mechanisms on the shared link were used. A DropTail queue produces rather correlated loss and causes round-trip time variations because of the queue length growth. A RED queue drops packets randomly and tries to keep the queue fairly short.

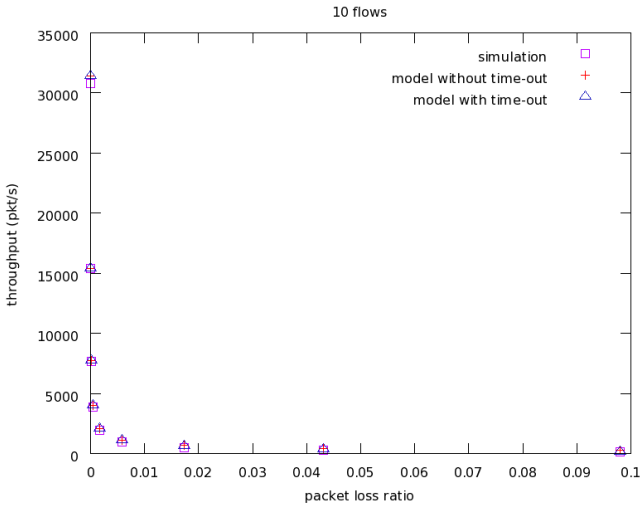
First we are going to present the results of the simulation in which a DropTail queue was used on the shared link. The queue length was set to the bandwidth $\times$ delay product. All access links had a delay of 1 ms. The capacity of the access links was 1 Gbps. The bottleneck delay was 30 ms and the bottleneck capacity was changed to cause a varying amount of loss. It had the values 1 Mbps, 2 Mbps, 4 Mbps, 8 Mbps, 16 Mbps, 32 Mbps, 64 Mbps, 128 Mbps and 256 Mbps (with the smallest number of flows and largest capacity used in this set of simulations, the shared link was still a bottleneck). Notably, using different bottleneck capacities influences the RTT. Each simulation was run for 600 seconds and first 50 seconds were not used for throughput measurements to eliminate the influence of the startup phase.

Figures 3.12 and 3.13 show that our equation also works well in this scenario. For smaller loss rates, the equation gives a good estimate of the throughput. For the shared link capacity of 256 Mbps the error is 1.5% for both 5 and 50 flows, and 2% and 3% for 10 and 100 flows; this error increase for 10 and 100 flows can be due to a phase effect. The simulations with the shared link capacities between 128 Mbps and 4 Mbps show a error that slightly increases as the link bandwidth decreases but the error is less than 7% for 5 and 10 flows. For 50 and 100 flows the error is a bit higher because of a phase effect. With burst loss, the probability that multiple packets are lost in a loss event is higher, and therefore the assumption that we made for calculating the fast-retransmission phase (in a loss event just one packet is lost) is incorrect. Besides the time-out probability calculation, this is the second imprecise calculation that is responsible for this mismatch. To be able to capture such a situation more precisely, we also need information about real loss. In our second model, besides another refinement, we explore this idea too.

In Figure 3.12 our equation slightly overestimates the throughput of 5 flows for the shared link bandwidth of 1 Mbps. In the 10 flows case, the throughput is overestimated for the 1 Mbps and 2 Mbps link bandwidth. But, in both cases, it can be noticed that flows experience a high loss rate. As in the previous simulation setup, this is probably due to the suboptimal time-out probability calculation. Figure 3.13 shows a similar result. We can see that our model slightly overestimates the throughput for 50 and 100 flows and a small link bandwidth, but it should be noted that the loss rate is more than 20%. Because of the large number of flows existing in the network, a strong phase effect occurred and it caused starvation of some connections.

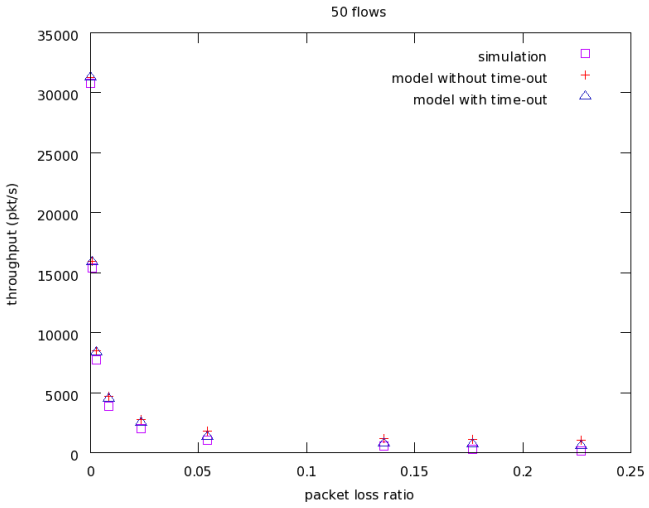


(a)

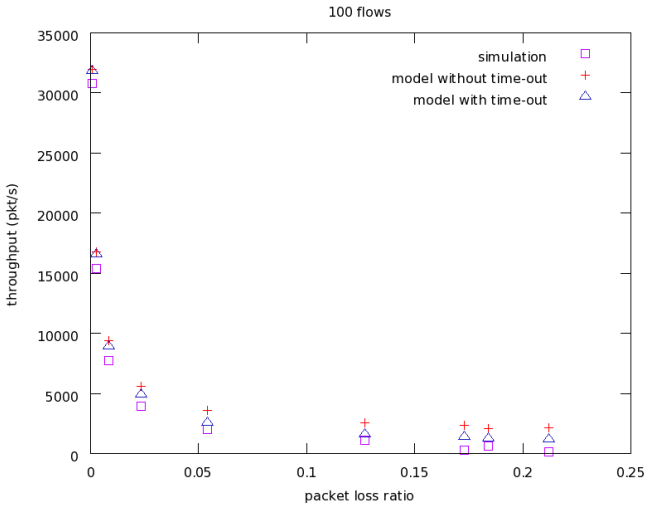


(b)

Figure 3.12: The model with per-flow loss measure: without any loss model, Drop-Tail queue, 30ms bottleneck delay



(a)



(b)

Figure 3.13: The model with the per-flow loss measure: without any loss model, DropTail queue, 30ms bottleneck delay

For making this effect weaker, a larger queue must be used, but that would be opposite to the accepted bandwidth × delay policy. Phase effects can not be eliminated completely in this way. They are one of the main disadvantages of DropTail queues. Since DropTail queues are present in the Internet, we wanted to evaluate our model also in such a scenario.

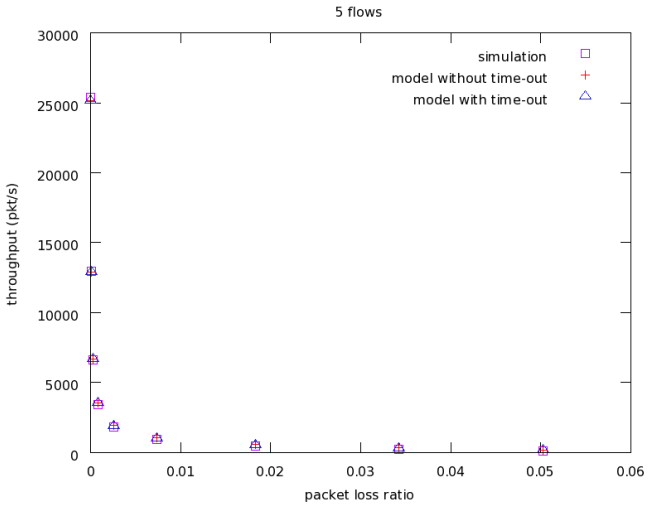
A RED queue influences the behavior of TCP in quite a different way than a DropTail queue. At the same time, the influence of this queuing mechanism is not the same as in the case of an artificially produced random loss. With a RED queue and without an additional loss model, the round-trip time on the path can vary. In the simulation with the random loss model, presented above, the bandwidth of the shared link and loss rates were chosen in a way that flows could never exceed the available capacity, so the round-trip time did not vary (although slight queuing delays, caused by packets being sent in bursts which is due to the way the ns-2 simulator operates, could have produced some round-trip time variations; the chosen link capacity was quite high, 1 Gbps, so the time needed for a packet to be transmitted was very short, less than 0.0078 ms, and thus, the delay caused by a small number of packets being enqueued was rather short).

The simulations setup was the same as with the DropTail queuing mechanism. The duration of each simulation was also the same. The RED parameters recommended in [3] were used: we let the algorithm implemented in ns-2 automatically configure them — $q\_weight = -1$, $thresh = 0$, and $maxthresh = 0$ were set for automatic configuration and the "gentle" mode was used. The bottleneck link capacity was varied as in previous subsection and it was set to: 1 Mbps, 2 Mbps, 4 Mbps, 8 Mbps, 16 Mbps, 32 Mbps, 64 Mbps, 128 Mbps and 256 Mbps.

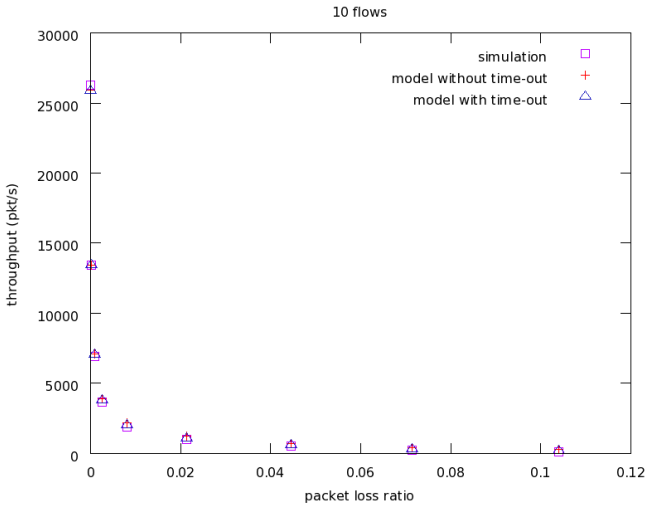
The results are shown in Figures 3.14 and 3.15. Our model estimates the throughput of flows traversing a RED queue even better than the throughput of flows sharing a DropTail queue. In the simulations with a DropTail queue time-outs and loss burst occurred more frequently than in these simulations. Therefore, for a higher loss rate, in all setups (with 5, 10, 50 and 100 flows), the difference between the measured throughput and its estimate using our equation is smaller than for the simulations with DropTail queue. For lower loss rates and a RED queue, the growth of the queue size is not as extreme as with a DropTail queue, and consequently, RTT variations were smaller. In this range as well, the throughput of flows traversing RED queue is better estimated than in the simulations with DropTail queue.

Simulations with burst loss

This set of simulations studied the accuracy of the model in case of a burst loss pattern. We used the same topology as in the simulation with random loss, but in this case, we used a loss model which caused random appearances of loss bursts. Loss is produced by a two-state loss model. It consists of the "good" state and the "bad" state; in the good state no packet is dropped, and when the loss model is in the bad state, all packets are dropped. The good state had a random duration and the probability that a packet ended this state was 0.001 (in our tests, this value had no significant influence on the accuracy of our model). The bad state had a constant duration which was varied from one simulation to the next. The burst, in other words the bad state duration, had values: 2 (always two packets were lost), 3 (always three

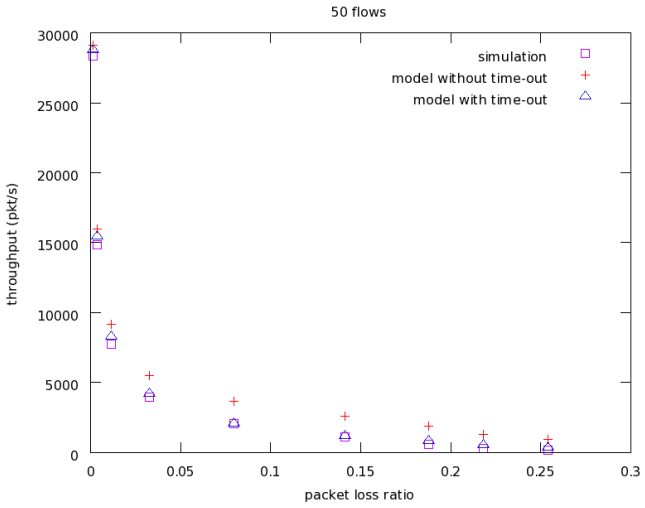


(a)

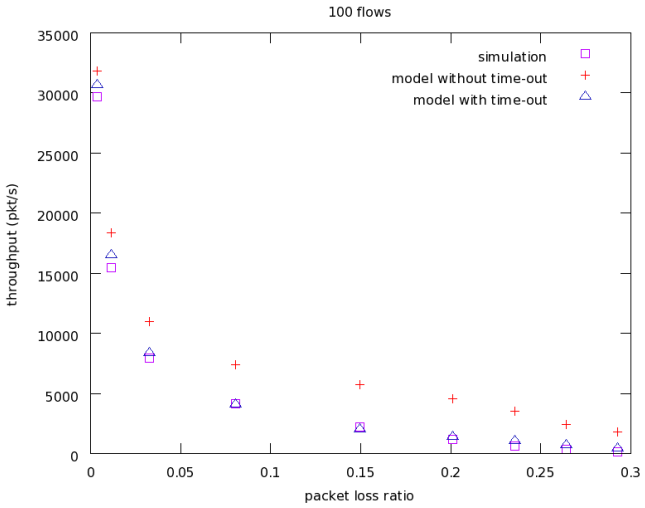


(b)

Figure 3.14: The model with per-flow loss measure: without any loss model, RED queue, 30ms bottleneck delay

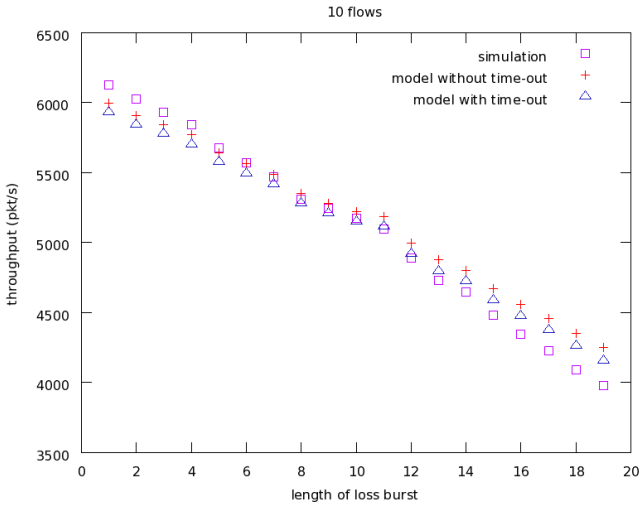


(a)

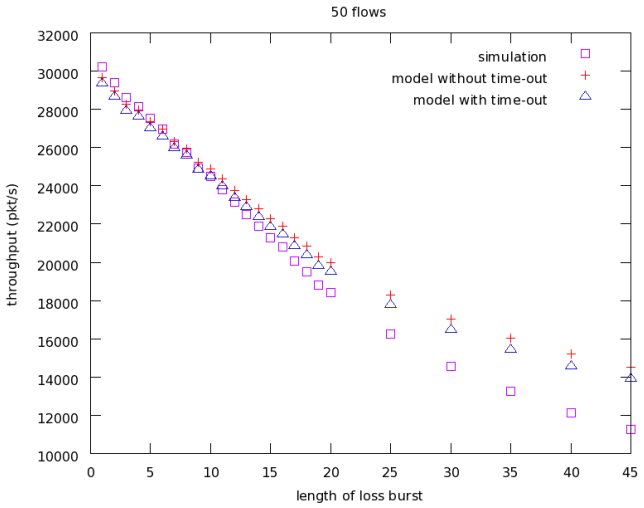


(b)

Figure 3.15: The model with the per-flow loss measure: without any loss model, RED queue, 30ms bottleneck delay

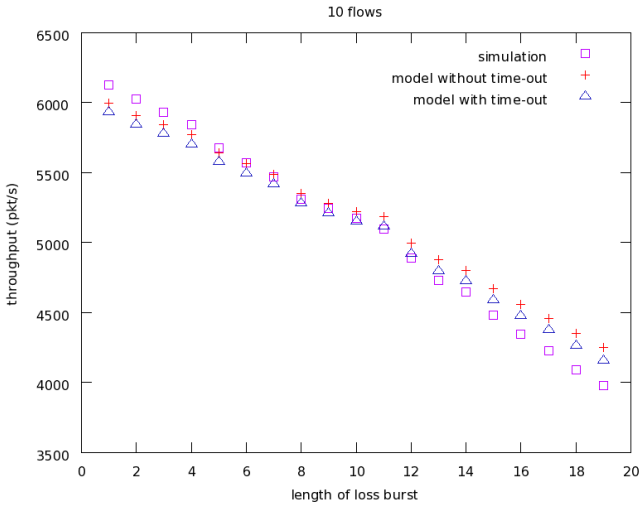


(a)

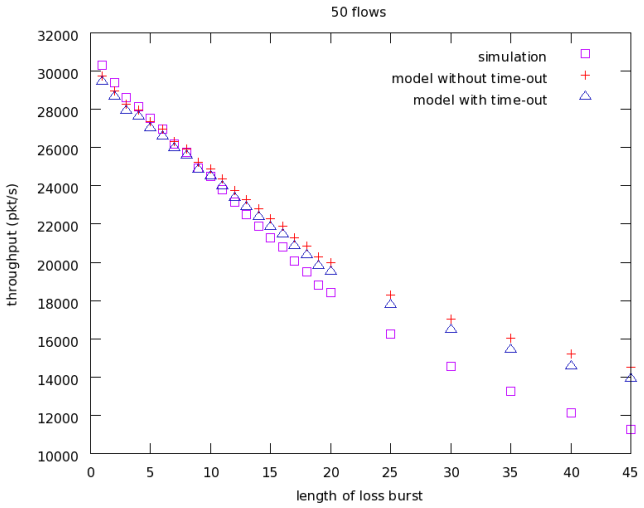


(b)

Figure 3.16: The model with the per-flow loss measure: burst loss, RED queue, 30ms bottleneck delay



(a)



(b)

Figure 3.17: The model with the per-flow loss measure: burst loss, DropTail queue, 30ms bottleneck delay

packets were lost), etc. The bottleneck queue used the RED queuing mechanism as well as the DropTail queuing mechanism. Each simulation lasted 600 seconds and also here the first 50 second of each simulation were neglected.

Figures 3.16 and 3.17 show a comparison between the throughput achieved by flows in the simulations and the throughput estimate obtained using our model. For the same reason already explained in the case of the simulation with random loss, the results of the simulations with a DropTail queue on the shared link (Figure 3.17) are almost identical as when a RED queue was used (Figure 3.16).

Our equation estimates the throughput quite well in the simulation setup with burst loss. In all simulations, the measured throughput and our model differ less than 4.5%. Also in these simulations, this mismatch is due to an error produced by the applied time-out probability calculation. This time-out probability calculation method was chosen because it gives in average the best results in a broad range of network conditions.

The difference between our model and the simulations is especially visible for a longer duration of bursts. Even though the "overhead_" parameter of the ns-2 simulator (setting this parameter causes each packet to be delayed at the sender for a random time factor) was used, mixing of packets belonging to different flows disappears at the beginning of a simulation. Therefore, in each loss burst almost only packets belonging to a single flow were dropped, and this caused time-outs to appear more frequently. The model without additional information about the average duration of loss bursts could not calculate the probability that a loss event is a time-out loss indication accurately enough.

3.3 The model — cumulative flow loss measure

The equation from Section 3.2 estimates the throughput of n parallel flows sharing the same path, and for this calculation the probability for any flow among these n flows to experience loss is needed. To calculate this probability, loss needs to be measured on a per-flow basis. Since, as already discussed, the complexity of obtaining this loss event rate measure is higher, we now explore the possibility of having the same calculation, but using the loss event rate of the cumulative flow.

Since, in a loss event of the cumulative flow one or more individual flows can receive a loss event indication, the loss event rate of the cumulative flow is lower than the loss event rate measured on a per-flow basis (the reader is referred to Section 3.1 for a more thorough discussion). Using the loss measure of the cumulative flow as a per-flow loss measure is equal to assuming that, in each loss event of the cumulative flow, just one individual flow experiences loss. This assumption rather deviates from reality as the number of observed flows and the loss rate grow

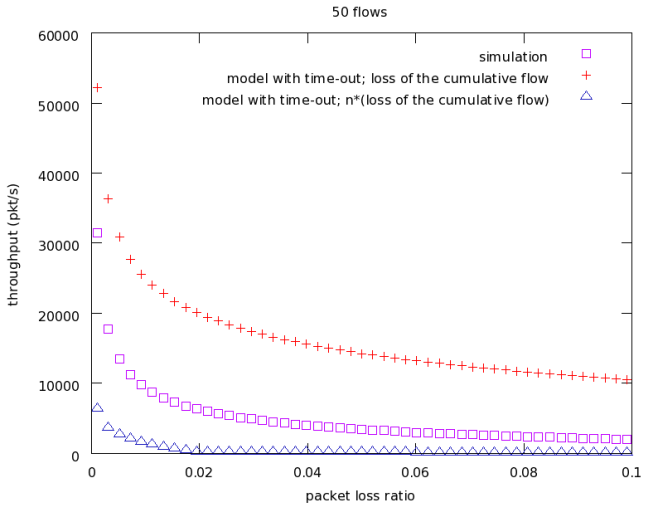


Figure 3.18: The equation from Section 3.2.2 and the loss event probability of the cumulative flow

(in simulations used for Figure 3.6, with only 5 flows and a loss rate of 0.9%, the average number of flows experiencing loss event indications in a loss event of the cumulative flow was 1.42 and, with 10 flows and the same loss rate, it was 1.96). Having this in mind, the results presented in Figure 3.18 are not surprising. It shows a comparison of the measurements and the results produced by the equation from Subsection 3.2.2, using the loss event rate of the cumulative flow instead of the loss event rate measured by observing each flow individually, as in Figure 3.7 (a). As it can be seen, the equation from Section 3.2.2 overestimates the throughput.

The second possibility would be to multiply the loss measure of the cumulative flow with the number of flows (n). Such a transformed loss measure would be an approximation of the per-flow loss measure with an assumption that, in each loss event of the cumulative flow, each individual flow experiences loss. This is not always correct as shown by the example above. In Figure 3.18, the third line shows this case (the equation from Section 3.2.2 and the loss measure of the cumulative flow multiplied by n). The model used in this way underestimates the measured throughput in all simulations.

In reality, the number of flows experiencing loss event indications in a loss event of the cumulative flow is between these two extremes and therefore these two ways of applying the model from Section 3.2 are not giving satisfying results, and the model needs to be changed so that it can be used when just the loss event rate of the cumulative flow is known.

In this extended equation we will include not only knowledge about the loss event rate but also knowledge about the real loss rate (the rate at which packet loss occurs). In a loss event of the cumulative flow, one or more flows can experience loss, and here we use information about

the real loss rate for estimating how many flows experience loss in one such a loss event. It should be noted that the number of flows experiencing loss in a loss event could be estimated using only the loss event rate. In that case, a certain distribution must be chosen and only the throughput of flows experiencing loss patterns which fit this distribution could be estimated well. We believe that using the real loss rate gives a greater flexibility and wider operating range. As already mentioned, knowledge about how many packets are lost in a loss event would enable a more precise calculation of the time-out probability. Having this extra information leads us to further refine the time-out part of our equation.

For the extended equation we take p_e to be the loss event rate of the cumulative flow, where one or more flows can experience loss. With p_r we denote the rate at which a packets (belonging to any flow) are lost.

3.3.1 Model considering only the congestion avoidance phase

In this section, as in Section 3.2, we observe the development of the window size over a TD-period, where in this case a TD-period is a period between two loss events of the cumulative flow. For the i-th 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. As in Section 3.2, we will derive the expression for B starting from (3.1):

$$B = \frac{E[Y]}{E[A]} \tag{3.35}$$

where $E[Y]$ is the average number of packets sent in a TD-period of the cumulative flow and $E[A]$ is the average duration of a TD-period.

The expression for calculating the average duration of a TD-period, expression (3.4), is not modified. Equation (3.7), the equation for the number of packets sent in a TD-period is almost the same. In this case we calculate the number of packets sent between two loss events of the cumulative flow and therefore, we use the loss event rate of the cumulative flow (p_e). We have:

$$E[A] = E[X]RTT \tag{3.36}$$

and

$$E[Y] = \frac{1}{p_e} \ . \tag{3.37}$$

Since we now consider loss events of the cumulative flow, in a loss event more than one flow can experience loss. In Section 3.2 we assumed that in each loss event just one flow was hit and the probability that a flow was hit in a TD-period was $1/n$. Let j_i be the number of flows belonging to the cumulative flow that experience loss at the end of the i-th TD-period. Assuming that loss is identically distributed over all flows, the probability that a flow is hit in

the i-th TD-period is j_i/n. To calculate the probability that there are k TDP between two loss events of an individual flow we take the same assumption as in Section 3.2 (Equation 3.2), but with a different flow hit probability, and we have:

$$P[loss\ in\ the\ k-th\ TDP] = \frac{j_i}{n}\prod_{l=1}^{k-1}(1-\frac{j_{(i-l)}}{n})\ . \tag{3.38}$$

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

$$P[A_f = kE[A]] = \frac{j}{n}(1-\frac{j}{n})^{k-1}\ . \tag{3.39}$$

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

$$\begin{aligned} E[A_f] &= \sum_{k=1}^{\infty}\left(\frac{j}{n}\left(1-\frac{j}{n}\right)^{k-1}kE[A]\right) \\ &= \left(\frac{nE[A]}{j}\right)\ . \end{aligned} \tag{3.40}$$

For the i-th TD-period let flows $\{m^e\}, e = 1..j_i$ (subset of n flows) be the j_i flows experiencing loss at the end of the period. As in Section 3.2 we will observe the evolution of the window size of an individual flow. Since we do not change any assumption about individual flows, from (3.8), the evolution of the window size of flow m^e between two loss events of the flow is:

$$W_{f_{(m^e)_{i_s}}} = \frac{W_{f_{(m^e)_{i_{s-1}}}}}{2} + \frac{X_{f_{(m^e)_{i_s}}}}{b} \tag{3.41}$$

where $W_{f_{(m^e)_{i_s}}}$ is the window size of flow m^e at the end of the (i_s)th TD-period, $X_{f_{(m^e)_{i_s}}}$ is the number of rounds from the end of (i_{s-1})th TD-period till the end of (i_s)th TD-period and $\{i_s\}, s = 1, 2...$ is the subset of TD-periods of the cumulative flow in which flow m^e experience loss event indications.

The window sizes of flows $\{m^e\}, e = 1..j_{i-1}$ are halved at the end of TDP_{i-1} and the window size of the cumulative flow at the beginning of the next TD-period is: $W_{i-1} - \sum_{e=1}^{j_{i-1}}\frac{W_{f_{(m^e)_{i-1}}}}{2} + (n - j_{i-1})$, where $W_{f_{(m^e)_{i-1}}}, e = 1..j_{i-1}$ are the window sizes of these j_{i-1} flows; the remaining $(n - j_{i-1})$ flows increase their window size by 1. Accordingly, we change equation (3.9) and now the number of packets sent by all flows in the i-th TD-period is:

$$\begin{aligned} Y_i = r_{i-1} \quad & + \\ \sum_{k=0}^{X_i/b-1} \quad & \left(W_{i-1} - \sum_{e=1}^{j_{i-1}} \frac{W_{f_{(m^e)_{i-1}}}}{2} + (n - j_{i-1}) + nk \right) b \\ & - r_i \ . \end{aligned} \tag{3.42}$$

Since the assumptions about individual flows are not altered, the steady state equation for an individual flow (as equation 3.10) is:

$$E[W_f] = \frac{2}{b} E[X_f] \ , \tag{3.43}$$

and from (3.36) and (3.40) we have:

$$E[X_f] = \frac{nE[X]}{j} \ . \tag{3.44}$$

In a similar way as in Section 3.2, at the end of a TD-period j flows have the window size $E[W_f]$, and other j flows that experience loss in the previous loss event have the window size $\frac{E[W_f]}{2} + \frac{E[X]}{b}$, $\frac{E[W_f]}{2} + 2\frac{E[X]}{b}$, etc. The mean value of the window size of the cumulative flow is:

$$E[W] = jE[W_f] + \sum_{k=1}^{\frac{n}{j}-1} j \left(\frac{E[W_f]}{2} + \frac{kE[X]}{b} \right) \ . \tag{3.45}$$

From (3.43), (3.44) and (3.45) we have:

$$E[W] = \frac{nE[X]}{2b} + \frac{3n^2 E[X]}{2bj} \ . \tag{3.46}$$

With the same assumptions as in Section 3.2, from (3.42), we have:

$$E[Y] = \sum_{k=0}^{E[X]/b-1} \left(E[W] - j\frac{E[W_f]}{2} + (n - j) + nk \right) b \tag{3.47}$$

and including (3.43), (3.44), (3.46) and (3.37):

$$\frac{1}{p_e} = \frac{3n^2 E[X]^2}{2bj} + \frac{nE[X]}{2} - jE[X] \ . \tag{3.48}$$

Solving this equation for $E[X]$we get:

$$E[X] = \frac{2j^2p_eb - np_ebj + \sqrt{(n^2p_e^2b^2j^2 - 4np_e^2b^2j^3 + 4j^4p_e^2b^2 + 24n^2p_ebj}}{6n^2p_e} \tag{3.49}$$

and including (3.46):

$$E[W] = \frac{2j^2p_eb - np_ebj + \sqrt{(n^2p_e^2b^2j^2 - 4np_e^2b^2j^3 + 4j^4p_e^2b^2 + 24n^2p_ebj}}{4bp_ej} \frac{2j^2p_eb - np_ebj + \sqrt{(n^2p_e^2b^2j^2 - 4np_e^2b^2j^3 + 4j^4p_e^2b^2 + 24n^2p_ebj}}{12bnp_e} . \tag{3.50}$$

From (3.35), (3.37), (3.36) and (3.49) we have:

$$B = \frac{1}{RTT} \frac{6n^2}{2j^2p_eb - np_ebj + \sqrt{(n^2p_e^2b^2j^2 - 4np_e^2b^2j^3 + 4j^4p_e^2b^2 + 24n^2p_ebj}} . \tag{3.51}$$

As already stated, according to [25], loss is usually not clustered (as stated: “single packet losses account for the large majority of the bursts, around 83%, and double losses occupy 12% of the bursts”), 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 time-outs and less by fast retransmissions; so this assumption could introduce just a minor error. Therefore we take the approximation: $j = \frac{p_r}{p_e}$. Since j must be less than n, we have $j = min(n, \frac{p_r}{p_e})$.

It is possible to relax this assumption by incorporating the possibility that more than one packet belongs to the same flow. For this case, we assume that a packet belongs to any of n flows with the same probability ($\frac{1}{n}$). The probability that a flow is not affected in a loss event (p_{n-1}) is equal to the probability that all $\frac{p_r}{p_e}$ lost packets belong to the other $n-1$ flows:

$$p_{n-1} = (\frac{(n-1)}{n})^{\frac{p_r}{p_e}} .$$

The probability p_1 that the flow is affected is:

$$p_1 = 1 - (1 - \frac{1}{n})^{\frac{p_r}{p_e}} .$$

The number of flows affected in a loss event with $\frac{p_r}{p_e}$ lost packets is:

$$j = max(n * (1 - (1 - \frac{1}{n})^{\frac{p_r}{p_e}}), 1) \ . \tag{3.52}$$

In the rest of the thesis we will refer to this j approximation as the improved j calculation.

The improved j approximation does not change significantly the equation result for most of our simulation setups. Therefore it will be used just in cases when the difference is visible. If not stated otherwise the $j = min(n, \frac{p_r}{p_e})$ approximation is used.

3.3.2 Model with time-outs

To include time-outs in the extended equation we will apply the same principle as in Section 3.2.2. Taking expressions (3.20) and (3.21), we get the steady state throughput equation with time-outs:

$$B_{ext} = B \ \frac{n - E[nTO]}{n} + E[nTO] \ \frac{E[R]}{E[Z^{TO}]} \tag{3.53}$$

where $E[R]$ and $E[Z^{TO}]$ are defined in (3.25) and (3.26) (only instead of p we have p_e), and $E[X]$ and B in (3.49) and (3.51) respectively. Since now we also include the information about real loss, the expression that gives the number of flows in the slow start phase ($E[nTO]$) differs. Let $E[nTO']$ be the number of flows experiencing time-outs in a loss event. Including flows in the slow start phase due to receiving a time-out loss indication in the previous loss round, we have $E[nTO] = E[nTO'] \ \frac{E[Z^{TO}]}{E[X] \ RTT}$.

To derive $E[nTO']$, let $pLost_i$ be the number of packets lost in TDP_i and let l_i flows (flows $\{m^e\}, e = 1..l_i$) experience time-outs at the end of the TD-period. $W_{f_{(m^e)_i}}, e = 1..l_i$ are congestion window sizes of these flows at the end of TDP_i. We have $pLost_i \geq \sum_{e=1}^{l_i} W_{f_{(m^e)_i}}$. Assuming that $\{pLost_i\}$ and $\{W_{f_{(m^e)_i}}\}$ 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 time-outs in a loss event, we have: $pLost \geq l \ \frac{E[W]}{n}$ (here we assume that packets experiencing time-out lose the entire current window of data). Since flows experiencing a triple-duplicate loss indication lose much less packets than flows experiencing a time-out loss indication, we can approximate that the number of flows experiencing a time-out in a loss event is

$$E[nTO'] = min(\frac{n \ pLost}{E[W]}, n) \ .$$

In a loss event the average number of packets lost is $pLost = p_r/p_e$. The congestion window size of the cumulative flow ($E[W]$) is given in (3.50), and the number of flows in the slow start phase in a TD-period is:

$$E[nTO] \quad = \quad min\left(\frac{n\ \frac{p_e}{p_r}}{E[W]}, n\right)\ \frac{E[Z^{TO}]}{E[X]RTT}\ .$$

Using expressions for $E[W]$ from (3.46) and for $E[Z^{TO}]$ from (3.26), we have:

$$E[nTO] \quad = \quad min\left(\frac{n\ \frac{p_e}{p_r}}{\frac{nE[X]}{2b} + \frac{3n^2E[X]}{2bj}}, n\right)\ \frac{f(p_e)T}{(1-p_e)E[X]RTT} \tag{3.54}$$

where

$$f(p_e) = 1 + p_e + 2p_e^2 + 4p_e^3 + 8p_e^4 + 16p_e^5 + 32p_e^6$$

and we approximate $j = min(n, \frac{p_r}{p_e})$; as in [103], $f(p)$ can be approximated with $1 + 32p^2$.

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

$$B = \frac{n - E[nTO]}{n\ p_e\ E[X]\ RTT} + \frac{E[nTO]}{f(p_e)T} \tag{3.55}$$

where $E[nTO]$ is given in (3.54) and $E[X]$ is defined in (3.49).

3.3.3 Algorithm

Here we give the equation derived in section 3.3 in the form of an algorithm. The input parameters are:

- RTT — is the round-trip time in seconds.
- b — is the maximum number of packets acknowledged by a single TCP acknowledgement.
- p_e — is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.
- j — is the number of packets lost in a loss event.

- t_RTO — is the TCP retransmission time-out value in seconds.
- n is the number of TFRC flows that MulTFRC should emulate. It is a positive rational number.

and x, j_af, a, z and q are temporary floating point variables. The algorithm returns the throughput in $packet/s$. Also here the equation in the form of an algorithm is give and not an complete implementation is suggested.

Algorithm 2 The model with the cumulative flow loss measure: The throughput of n parallel flows [pkt/s]

$j\_af = \min(\max(j, 1), n)$;
$a = p\_e * b * j\_af * (24 * n^2 + p\_e * b * j\_af * (n - 2 * j\_af)^2)$;
$x = (j\_af * p\_e * b * (2 * j\_af - n) + sqrt(a))/(6 * n^2 * p\_e)$;
$q = min(2 * j * b/(x * (1 + 3 * n/j)), n)$;
$z = t_RTO * (1 + 32 * p\_e^2)/(1 - p\_e)$;
if $q * z/(x * RTT) \geq n$ **then**
  $q = n$;
else
  $q = q * z/(x * RTT)$;
end if
return $((1 - q/n)/(p\_e * x * RTT) + q/(z * (1 - p\_e)))$;

3.3.4 Validation

For validating the extended equation we used the same simulations as in section 3.2.4. Since simulation parameters were not modified and the same network topology ("dumbbell" topology) was used, we will not present them in detail here. For this version of the equation we need the loss event rate of a cumulative flow. We measured the loss event rate by counting just one loss event of individual flows per RTT. If one flow experienced a loss event at time t_{le_1}, loss events of other flows at time $t_{le'} \leq t_{le_1} + RTT$ were ignored. The same principle for calculating the number of loss events is used in [61].

As our validations show, the equation performs well with a broad range of loss probabilities. The equation slightly underestimates the throughput for a very small loss rate. The reason for this is the approximation for calculating j (the average number of flows experiencing loss in a loss event of the cumulative flow) as $j = \frac{p_r}{p_e}$, where we assume that each lost packet in a loss event belongs to a different flow. In reality more than one packet can belong to the same flow and, in these cases, the improved j approximation can give better results. In the following text, we will point out such cases.

Simulations with random loss

In this set of simulations we used the same setup as in the simulations with random loss presented in Subsection 3.2.4 (each packet is dropped with the same probability). We also used the DropTail and RED queuing mechanism on the shared link.

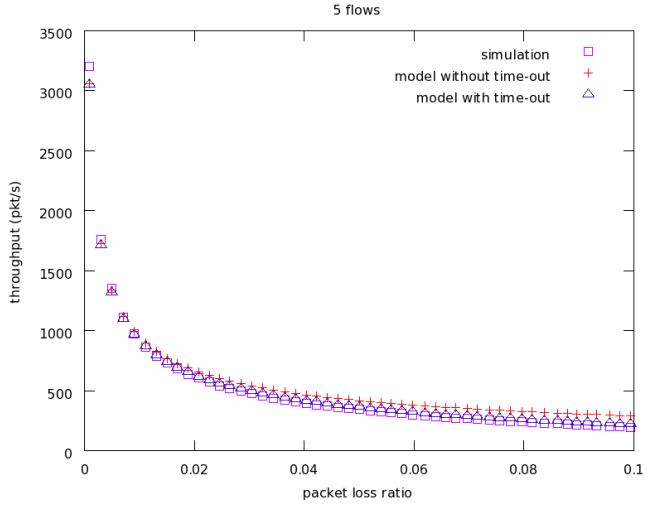
Figures 3.19 and 3.20 show validations with random loss, a RED queue and a delay of 30 ms on the bottleneck link. Further, Figure 3.21 shows that the equation also performs well with a longer RTT (the bottleneck delay was 100 ms).

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, lower than 0.01%. In that case, our equation underestimates the measured throughput, but the error is never greater than 5%.

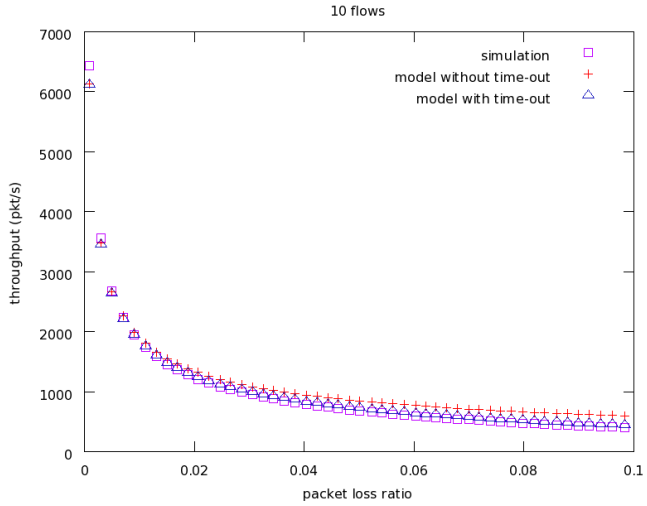
To remind the reader, we approximate j, the number of individual flows which experience loss events in a loss event of the cumulative flow, by the number of packets lost in a loss event of the cumulative flow. Even though, in the simulations with this low loss rate, the difference between the approximation and the measured j (the measured j is the average number of flows experiencing loss in a loss event of the cumulative flow; this is obtained looking at each flow separately) is quite small (e.g. the j approximation for the lowest loss rate shown in Figure 3.19 (a) was 1.175302; the measured j — the average number of flows experiencing loss in a loss event of the cumulative flow — was 1.124352), 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%). As the loss rate grows, the rate decreases and the error produced by the j approximation becomes smaller. As we mentioned, the improved j calculation will be used in some cases. For the described example, the improved j approximation gives a value of 1.153449 and the rate calculated using the improved j produces an error of 3.87% which is a better estimate than with the old j approximation. To measure j properly would however require each flow to be observed separately and this is what we wanted to avoid. Figure 3.21 shows the same results in this loss range.

In Figure 3.19, the error exceeds 7% for a loss rate greater than 4.5%. In the shown results, the differences never exceed 10.5%. For larger loss rates, larger than 3%, time-outs gain more influence on the throughput and in this part the time-out probability calculation is responsible for the mismatch; the fact that this error is a bit higher as the round-trip time increases, see Figure 3.21, confirms our suspicions. The time-out probability depends on $\frac{RTT}{T}$ (see Figure 3.54) and as RTT grows this value becomes smaller and decreases the time-out probability, but the correct value should be larger.

We also set the queue on the shared link to use the DropTail queue mechanism. The results are shown in Figures 3.22, 3.23 and 3.24. As already discussed in subsection 3.2.4, the similarity between simulations with RED and DropTail is to be expected.

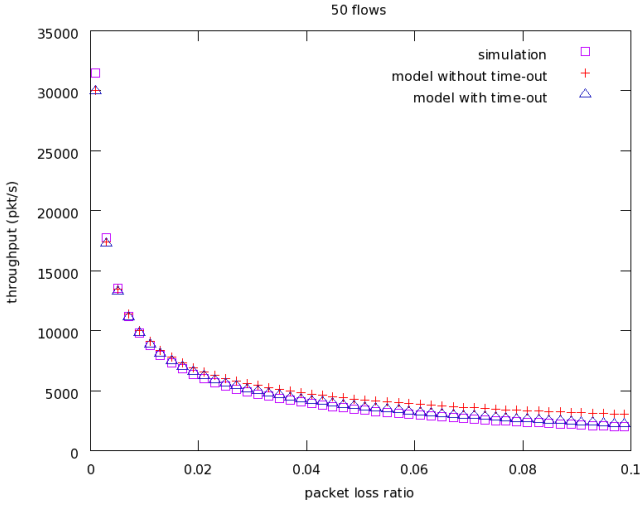


(a)

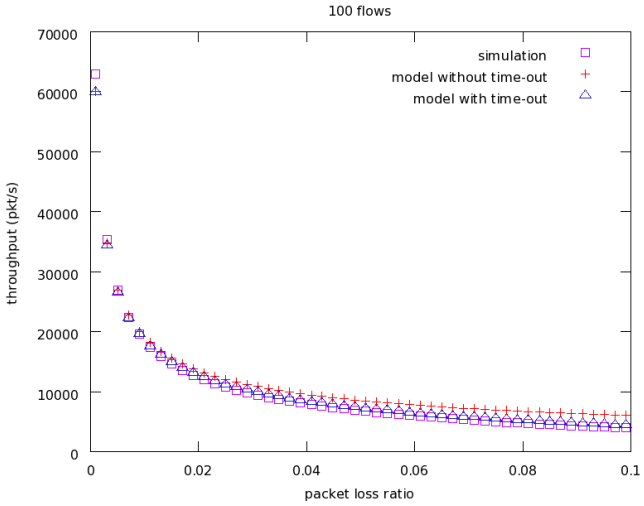


(b)

Figure 3.19: The model with the cumulative flow loss measure: random loss (each packet is lost with the same probability), RED queue and 30ms delay on the shared link

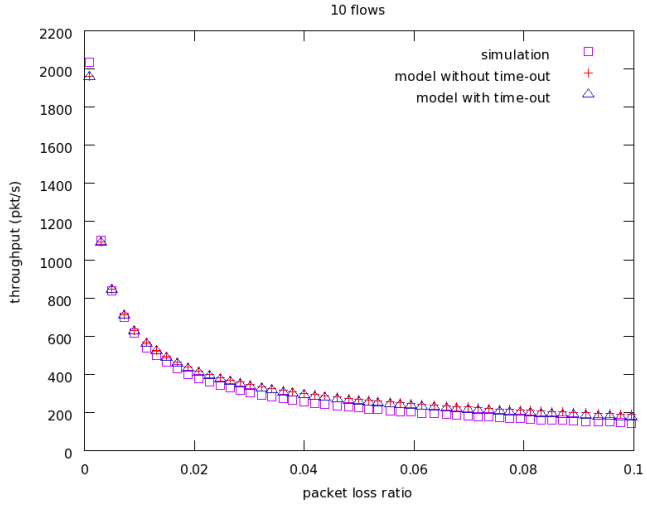


(a)

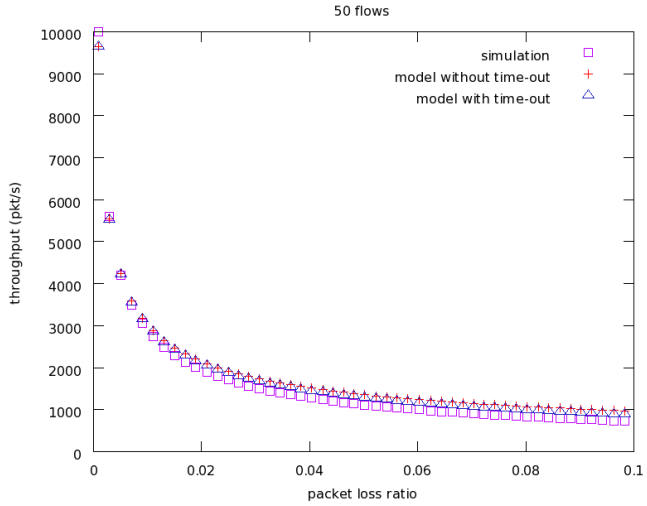


(b)

Figure 3.20: The model with the cumulative flow loss measure: random loss (each packet is lost with the same probability), RED queue and 30ms delay on the shared link

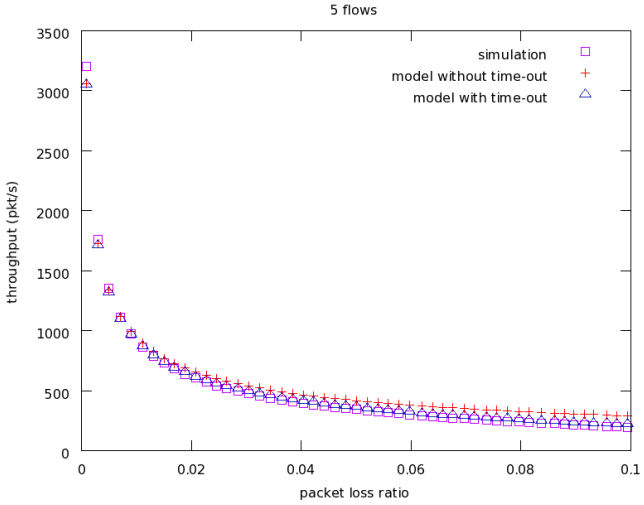


(a)

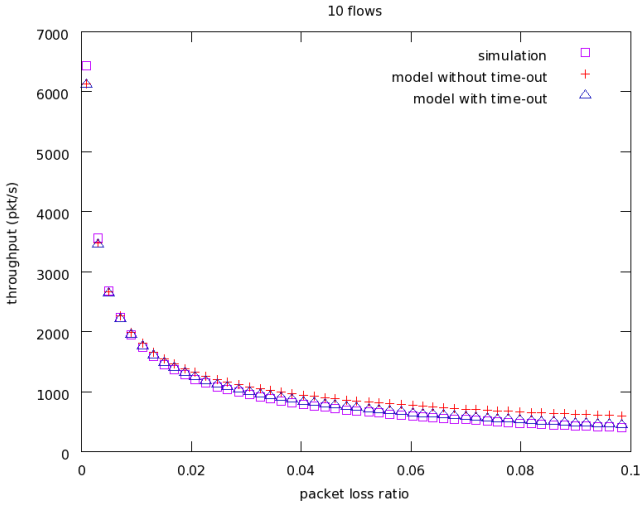


(b)

Figure 3.21: The model with the cumulative flow loss measure: random loss (each packet is lost with the same probability), RED queue and 100ms delay on the shared link

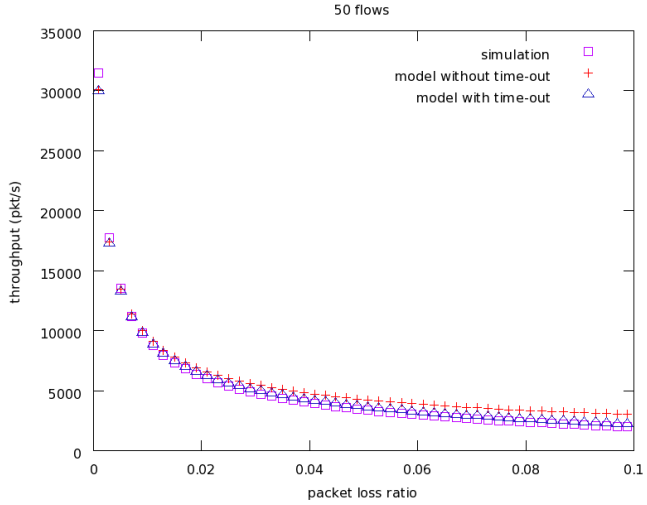


(a)

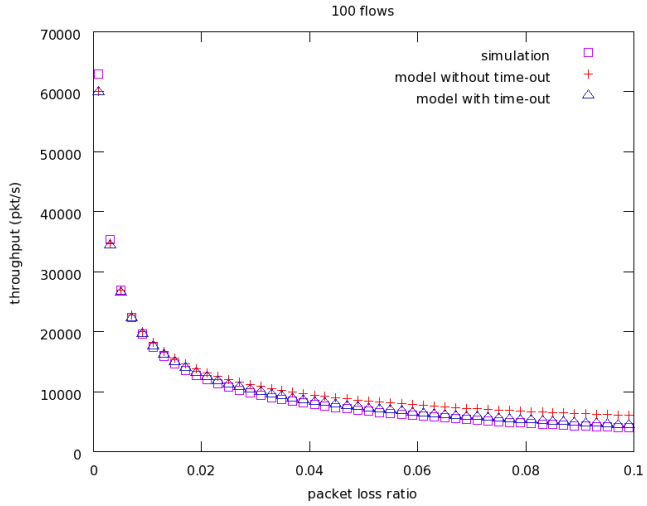


(b)

Figure 3.22: The model with the cumulative flow loss measure: random loss (each packet is lost with the same probability), DropTail queue and 30ms delay on the shared link

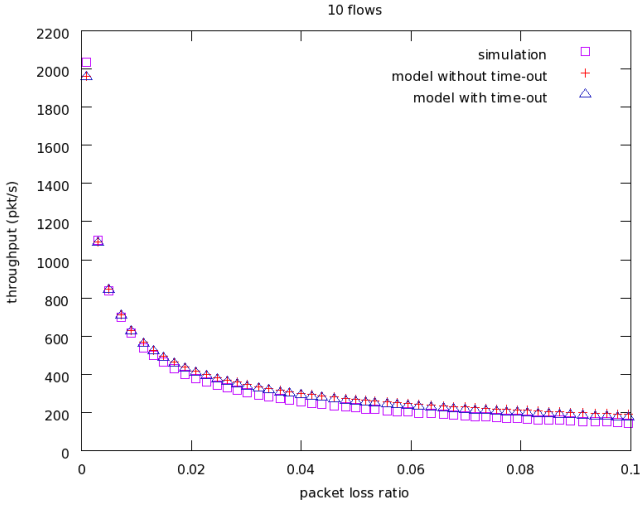


(a)

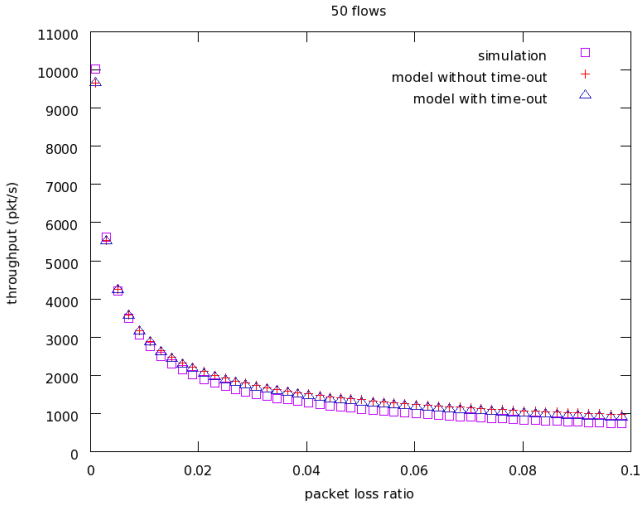


(b)

Figure 3.23: The model with the cumulative flow loss measure: random loss (each packet is lost with the same probability), DropTail queue and 30ms delay on the shared link



(a)



(b)

Figure 3.24: The model with the cumulative flow loss measure: random loss (each packet is lost with the same probability), DropTail queue and 100ms delay on the shared link

Simulations with loss from a DropTail (FIFO) queue and from a RED queue

The simulations presented in this subsection are the same as the simulations without any loss model in addition to loss from RED or DropTail queuing presented in Subsection 3.2.4, only here our model with the cumulative flow loss measure is compared with the simulations results.

We ran simulations with the DropTail queue mechanism on the shared link as well as the RED queue mechanism on the shared link. Figures 3.25 and 3.26 show results when a DropTail queue was used and Figures 3.27 and 3.28 show simulations with a RED queue. In all figures, it can be seen that our equation estimates throughput well.

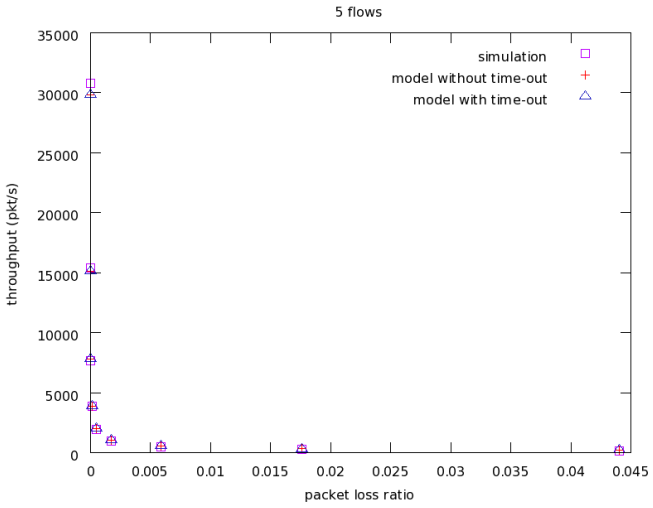
First, we will observe the simulations with a DropTail queue on the shared link. For the simulations with a higher capacity (256 Mbps and 128 Mbps) of the shared link the difference between the simulation and our equation was very small, even less than 1%, except for 5 flows where the error is around 3%; in this case the error is due to the use of the j approximation and, as the number of flows increases (in our simulations, already 10 flows), the j approximation does not introduce a significant error.

Figures 3.27 and 3.28 show results when a RED queue was used on the shared link. In the same way as for simulations with the DropTail queuing mechanism on the shared link, for a small number of observed flows (in this case 5 flows and 10 flows) and a lower loss rate the error is introduced by the j approximation (e.g. for the lowest lost rate and 5 flows, the measured j was 3.43859 and the j approximation had the value 4.140351; the measured throughput was 25403.79 pkt/s, the calculation with the j approximation gave 22903.79 and with the measured j it would have given a throughput of 25139.24; the improved j approximation is 3.015146 which is smaller than the measured j and the equation with the improved j overestimates the throughput; the estimate is 26850.08).

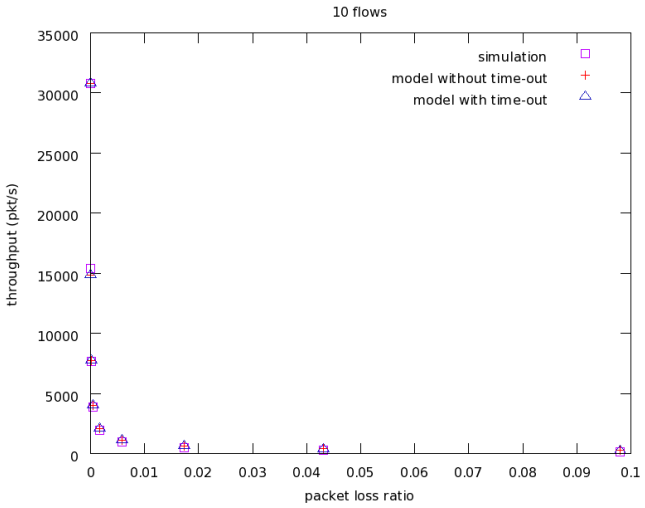
Our equation overestimates the throughput of 50 and 100 flows in case of a low shared link bandwidth and both queuing mechanisms (RED and DropTail); this is seen in Figures 3.26 and 3.28. Investigating further, we came to the conclusion that in these cases strong correlation between flows caused starvation of some connections. Even using a RED queue and the "overhead_" parameter, present in the ns-2 simulator (this option introduces an additional delay to each packet before it leaves the sender), to eliminate this phase effect [53] did not show an improvement. We believe the phase effect was still present because the queue on the shared link was too small for 50 and 100 flows; this effect could be eliminated by increasing the length of the buffer of the shared link which would be against the common bandwidth $\times$ delay product "rule" for a queue length and the automatic configuration for the RED queue mechanism could not be used.

Simulations with burst loss

The opposite extreme cases to the random loss that can occur in the network is burst loss. Therefore we validated the accuracy of our second model in the presence of burst loss with a controlled burst duration. The setup for this simulations is the same as in subsection 3.2.4 (the

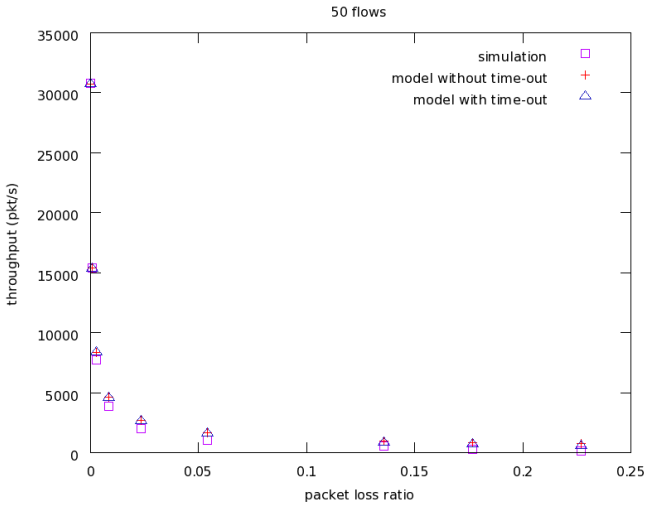


(a)

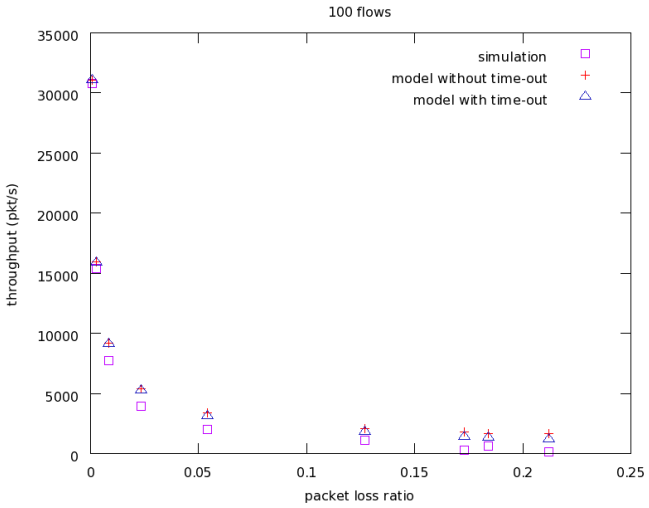


(b)

Figure 3.25: The model with the cumulative flow loss measure: without any loss model, DropTail queue, 30ms bottleneck delay

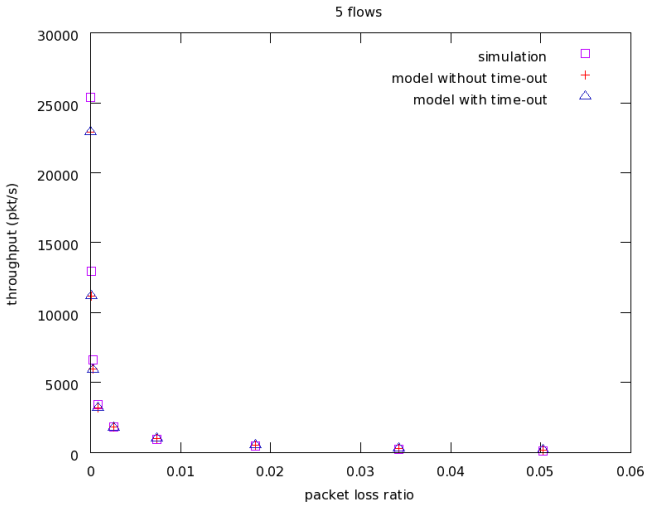


(a)

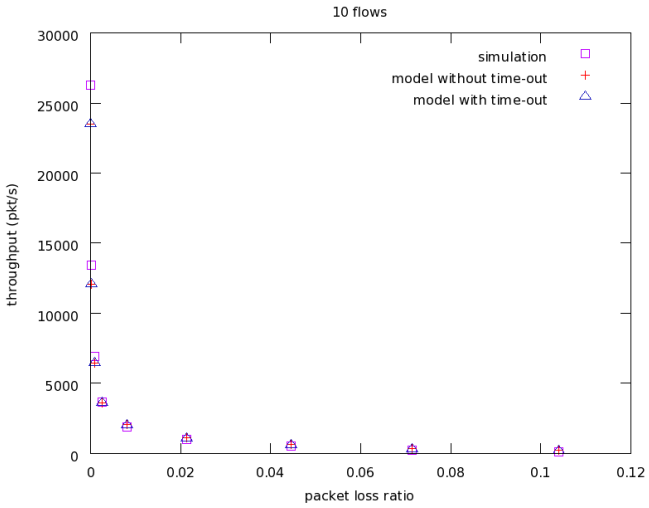


(b)

Figure 3.26: The model with the cumulative flow loss measure: without any loss model, DropTail queue, 30ms bottleneck delay

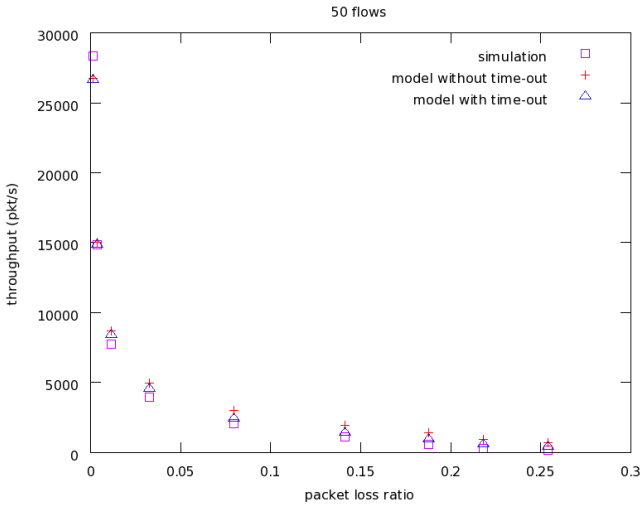


(a)

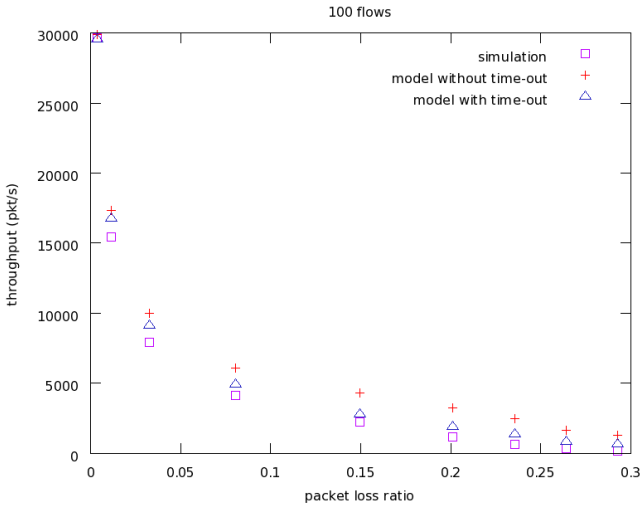


(b)

Figure 3.27: The model with cumulative flow loss measure: without any loss model, RED queue, 30ms bottleneck delay

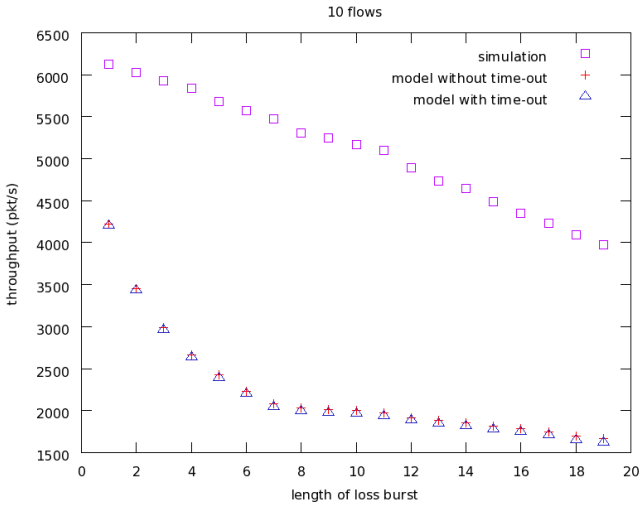


(a)

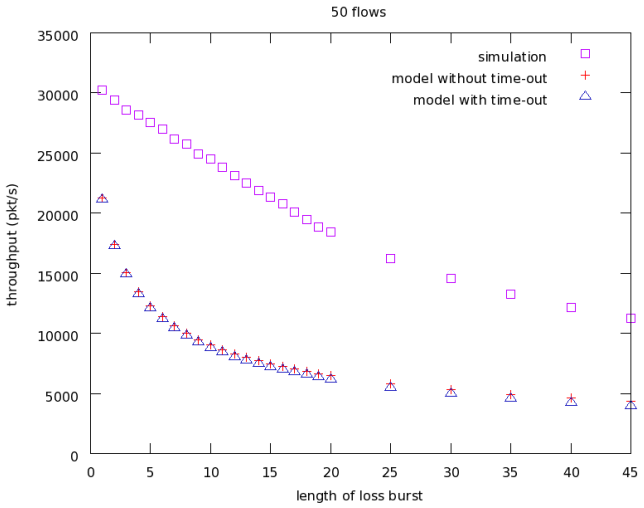


(b)

Figure 3.28: The model with cumulative flow loss measure: without any loss model, RED queue, 30ms bottleneck delay

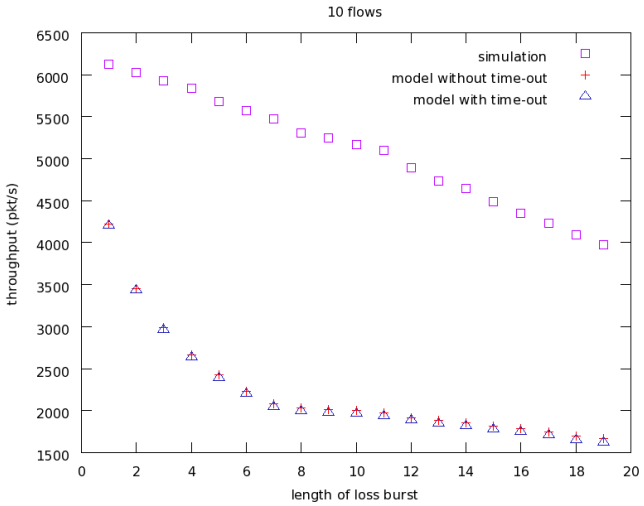


(a)

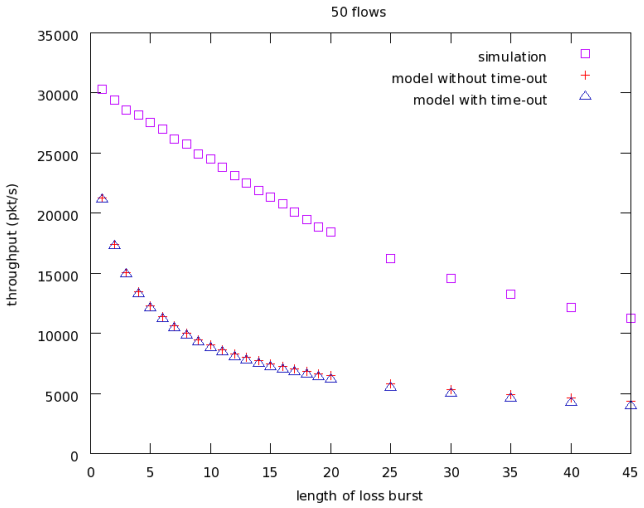


(b)

Figure 3.29: The model with the cumulative flow loss measure: burst loss, RED queue, 30ms bottleneck delay



(a)



(b)

Figure 3.30: The model with the cumulative flow loss measure: burst loss, DropTail queue, 30ms bottleneck delay

simulations with burst loss). Opposite to our first model, we expected that our second model would show fairly unsatisfying results. Some shortcomings of the ns-2 simulator are to be seen here, i.e. there is no multiplexing between flows. Even though the "overhead_" ns-2 parameter [53] was used the multiplexing between flows was not present. In other words packets from the same flow were always sent back-to-back. Such a extreme behavior is not to be expected in the Internet. As the burst loss model always dropped a number of consecutive packets, the dropped packets always belonged to the same flow. In each loss event a number of packets were dropped, but they were from the same flow; since our model assumes that packets are mixed (the first packet belongs to the first flow, the second packet belong to the second flow etc.) it interprets multiple losses in a loss event in the wrong way. Therefore, it always underestimates the measured throughput and, as the number of consecutively dropped packets increases, the error grows (Figures 3.29 and 3.30). Since this effect is very strong even the improved j calculation could not give satisfying results. The equation with the per-flow loss measure takes a more precise measure of loss events of individual flows; due to this knowledge it can more accurately estimate the throughput of observed flows.

3.3.5 Real-life measurements

We also validated our extended equation with real-life measurements. The measurement were run in two time periods. The first set of measurements were carried out in spring 2008, but because of operational restrictions at the University of Innsbruck, the network card bandwidth was limited and proper measurements of the network with a bottleneck inside the network could not be performed as explain in more details below. Therefore, in autumn 2009 a second set of measurements were carried out. Since the university policy changed, for these measurements, there were no restrictions on our access link.

The first set of measurements

Obtaining measurements of parallel flows is a challenging problem. The window scaling option is not widely used (because of certain routers not supporting it), and having high link capacities (as in the case of a university connection), TCP flows are mostly receiver limited and a non-limited steady-state behavior of the congestion control cannot be produced. Therefore, at the time the first set of measurements were carried out, the use of PlanetLab [2] (the window scaling option was enabled, but changing receiver buffer size was not allowed) or measuring throughput of parallel downloads from a web site were not possible. For our measurements we used hosts situated at two university sites[1], however not without minor problems.

We measured the throughput of $n = 1..10$ connections opened between two hosts. All connections started at the same time. We sent data from our site to Texas and Ireland and used Web100[2], installed on our site, for monitoring.

[1] We would like to thank Andrea Fumagalli and Doug Leith for letting us access their sites for measurements
[2] http://www.web100.org/

On our site the host (Athlon64 3200+, 512 MB RAM, 100 Mbit network card) had a Linux kernel version 2.6.17.1 and Web100 version 2.5.11. Because of almost no loss in the network we would have needed to transfer files of 1-2 GB to get sustained steady state TCP behavior, which we could not do because of operational restrictions at the University of Innsbruck. Therefore we set the network card to work with only 10 Mbps. Without a limiting network card, it is possible to achieve a transfer with a higher rate than when this limit is in place. Therefore, we assume that, in this set of measurements, loss mostly occurred in the access link.

Innsbruck — Ireland

The host in Ireland (149.157.192.252) ran Linux and the following TCP parameters were set:

- window scaling was turned on, and the advertised window scaling value was 12
- SACK was enabled

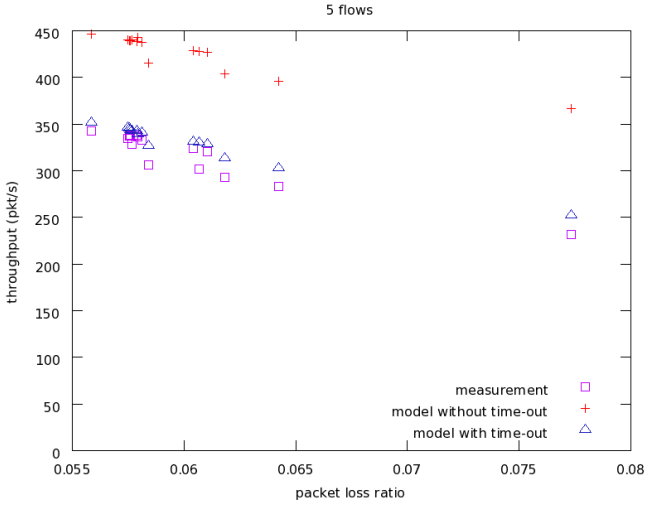
The measurements took from the 30th of May 2008 (16:17) till the 1st of June 2008 (17:28). We transmitted files of 70 MB using HTTP, opening n (1..10) uploads at the same time. For each number n, the measurement was run multiple times. Every transfer lasted at least 700 s (up to 1300 s). Since a cumulative flow rate was always less than 10 Mbps and the measurements were fairly long, the initial slow start phase could not have large impact on the throughput measurements; therefore a measuring period began at the same time as a transfer. The traceroute to this host is listed below:

```
 1  192.168.64.1 (192.168.64.1)  0.312 ms  0.395 ms  0.233 ms
 2  sr01a.uibk.ac.at (138.232.65.126)  0.463 ms  0.472 ms  0.449 ms
 3  rborder.uibk.ac.at (138.232.15.222)  3.639 ms  2.297 ms
    0.983 ms
 4  Ibk.ACO.net (193.171.19.1)  1.070 ms  0.740 ms  0.562 ms
 5  Wien2.ACO.net (193.171.12.209)  9.859 ms  10.099 ms  10.081 ms
 6  Wien21.ACO.net (193.171.23.22)  10.332 ms  10.347 ms  10.328 ms
 7  aconet.rt1.vie.at.geant2.net (62.40.124.1)  10.340 ms
    10.458 ms 10.211 ms
 8  so-7-0-0.rt1.pra.cz.geant2.net (62.40.112.6)  16.823 ms
    17.091ms  16.699 ms
 9  so-6-3-0.rt1.fra.de.geant2.net (62.40.112.38)  24.955 ms
    24.838ms  24.688 ms
10 so-5-0-0.rt1.ams.nl.geant2.net (62.40.112.58) 32.074 ms
    32.332ms  32.194 ms
11 so-4-0-0.rt1.lon.uk.geant2.net (62.40.112.138) 40.448 ms
    40.203ms  40.310 ms
12 62.40.125.126 (62.40.125.126) 51.443 ms 51.084 ms  50.874 ms
13 gige0-1-937-nuim1.cwt.client.hea.net (193.1.236.66)  51.563 ms
    51.535 ms  51.675 ms
14 149.157.1.201 (149.157.1.201) 52.067 ms 51.545 ms  51.309 ms
15 149.157.1.11 (149.157.1.11) 52.188 ms 52.191 ms  52.060 ms
```

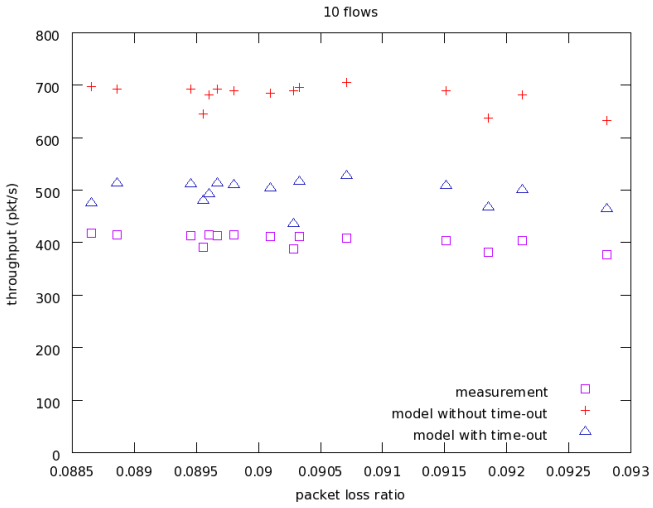
Innsbruck — Texas

The host in Texas (129.110.241.44) ran Linux and used the following TCP parameters:

- window scaling was turned on, and the advertised window scaling value was 6
- SACK was enabled

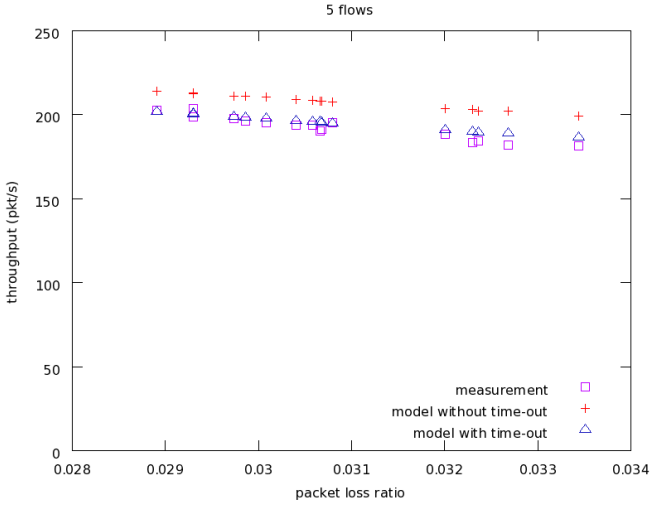


(a)

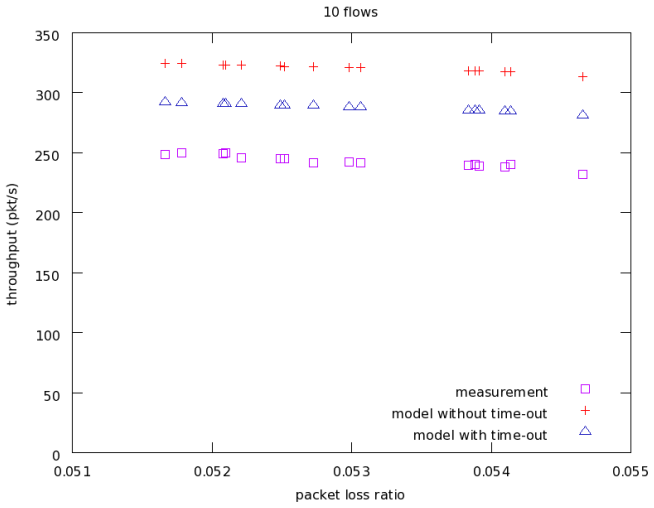


(b)

Figure 3.31: Measurements Innsbruck - Ireland, using the equation with the loss event rate of the cumulative flow



(a)



(b)

Figure 3.32: Measurements Innsbruck - Texas, using the equation with the loss event rate of the cumulative flow

We measured starting from the 9th of May 2008 (15:33) till the 13th of May 2008 (15:35). We performed the same set of HTTP file transfers as to Ireland. Each measurement took at least 800s (up to 2100s). The traceroute to this host is:

```
 1  192.168.64.1 (192.168.64.1)  0.283 ms  0.288 ms  0.748 ms
 2  sr01a.uibk.ac.at (138.232.65.126)  0.515 ms  0.908 ms  0.621 ms
 3  rborder.uibk.ac.at (138.232.15.222)  2.367 ms  0.831 ms
    2.147 ms
 4  Ibk.ACO.net (193.171.19.1)  1.511 ms  0.971 ms  2.701 ms
 5  Wien2.ACO.net (193.171.12.209)  10.053 ms  9.970 ms  10.207 ms
 6  Wien1.ACO.net (193.171.23.33)  10.218 ms  10.220 ms  10.332 ms
 7  vix2.core01.vie01.atlas.cogentco.com (193.203.0.113)  10.847 ms
    10.465 ms 10.460 ms
 8  te1-3.ccr01.muc01.atlas.cogentco.com (130.117.3.21)  17.203 ms
    17.073 ms 16.950 ms
 9  te8-3.mpd02.fra03.atlas.cogentco.com (130.117.0.165)  29.953 ms
    47.325 ms
    te1-1.ccr01.str01.atlas.cogentco.com (130.117.3.77)  19.212 ms
10  te2-1.ccr01.ams03.atlas.cogentco.com (130.117.2.141)  37.323 ms
    te7-1.mpd02.fra03.atlas.cogentco.com (130.117.3.81)  29.844 ms
    te2-1.ccr01.ams03.atlas.cogentco.com (130.117.2.141)  36.958 ms
11  te3-1.ccr01.ams03.atlas.cogentco.com (130.117.2.202)  37.080 ms
    te7-1.mpd02.lon01.atlas.cogentco.com (130.117.1.118)  45.207 ms
    te1-1.ccr01.ams03.atlas.cogentco.com (130.117.1.162)  37.447 ms
12  te2-1.ccr01.lon01.atlas.cogentco.com (130.117.1.170)  45.074 ms
    te4-1.ccr04.jfk02.atlas.cogentco.com (66.28.4.253)  114.048 ms
    te1-1.ccr01.lon01.atlas.cogentco.com (130.117.1.110)  45.541 ms
13  te2-2.mpd01.ord01.atlas.cogentco.com (154.54.6.18)  140.631 ms
    te3-2.mpd01.bos01.atlas.cogentco.com (130.117.0.185)
    119.275 ms te9-4.mpd01.ord01.atlas.cogentco.com (154.54.7.81)
    141.027 ms
14  te4-1.ccr04.jfk02.atlas.cogentco.com (66.28.4.253)  114.352 ms
    te9-4.ccr02.ord01.atlas.cogentco.com (154.54.7.169)  141.781 ms
    142.155 ms
15  te2-2.ccr02.bos01.atlas.cogentco.com (154.54.5.242)  118.511 ms
    te9-4.ccr02.mci01.atlas.cogentco.com (154.54.6.214)  153.746 ms
    te9-4.mpd01.mci01.atlas.cogentco.com (154.54.7.138)  152.756 ms
16  te3-4.ccr02.dfw06.atlas.cogentco.com (154.54.0.126)  161.841 ms
    te8-4.mpd01.dfw01.atlas.cogentco.com (154.54.5.125)  162.138 ms
    te7-4.ccr02.dfw01.atlas.cogentco.com (154.54.2.113)  248.601 ms
17  te3-4.ccr02.dfw06.atlas.cogentco.com (154.54.0.126)  162.229 ms
    te2-2.ccr02.mci01.atlas.cogentco.com (154.54.25.77)  151.646 ms
    te2-4.ccr02.dfw06.atlas.cogentco.com (154.54.0.138)  163.848 ms
18  utd.demarc.cogentco.com (38.104.34.26)  162.747 ms
    vl-431-utd-ntg-gw1.northtexasgigapop.org (208.76.224.74)
    158.259 ms utd.demarc.cogentco.com (38.104.34.26)  162.463 ms
19  utdgw2-vl5-ge-2-2.utdallas.edu (129.110.5.71)  157.882 ms
    vl-431-utd-ntg-gw1.northtexasgigapop.org (208.76.224.74)
    157.973 ms 159.261 ms
```

The loss event rate was measured in the same way as in case of the simulations (not more than one loss event per RTT). Both Figures (3.31 and 3.32) show that our equation also yields a reasonably good estimate of the throughput with real-life measurements. The measurements of 5 flows is accurately estimated by our equation. The throughput of 10 flows is slightly overestimated by our equation and the use of the improved j calculation would not show any improvement.

The second set of measurements

The second set of measurements was also carried out from the host situated in Innsbruck. As the circumstances changed, we were able to perform measurements without limiting the network card to only 10 Mbps, it had 100 Mbps. Secondly, the PlanetLab nodes were also

changed and now a user has been able to change some network settings using vsys. Vsys is a tool that provides privileged access to a system for non-privileged users similar as tools "sudo" and "Proper". It was first deployed in May 2008 (starting with PlanetLab Version 4.2).

We measured the throughput of $n \in \{10, 20\}$ connections open between two hosts. Measurement durations were varied and we ensured that the interval taken for the throughput measurements was long enough to enable capturing the steady-state behavior of the connections. For each connection, window scaling was enabled and the advertised window scaling value was large enough for the measured connections not to be limited by the receiver. We ensured that the connections were not sender limited as well.

The host in Innsbruck (138.232.65.6) ran Linux (kernel version 2.6.30) and Web100[3] version 2.5.25. To validate the equation with shorter as well as longer round-trip times, a couple of sites in Europe and outside Europe were chosen as the destination. The European sites were situated in: France (planetlab2.utt.fr), Italy (planetlab2.elet.polimi.it) and Portugal ((planetlab1.fct.ualg.pt). Two non-European hosts were situated in: Korea (csplanetlab2.kaist.ac.kr) and Uruguay (planetlab-1.fing.edu.uy).

The number of flows experiencing loss event indications in a loss event of the cumulative flow (denoted by j) is approximated by the number of packets lost in a loss event of the cumulative flow (in the following text we will refer to this j approximation as the old j approximation). At the end of subsection 3.3.1, we showed that j can be calculated in a different way too. This j approximation takes into account the possibility that more than one packet belongs to the same flow. We call this approximation the improved j approximation. For some of the following measurements, this change will be applied to the equation.

As it will be seen in the following text, the mismatch between the measured throughput and its estimate using our equation are caused by two factors: the error introduced by the j approximation and the error introduced by a mismatch between the average round-trip time and the average duration of a round (the time between two window increases, in other words the time needed for a flow to send its current window size of data).

Innsbruck — France

The first host was situated at Universite De Technologie De Troyes, Troyes, France (the host name: planetlab2.utt.fr — 194.254.215.12). This host ran PlanetLab software (kernel version $2.6.22.19 - vs2.3.0.34.39.planetlab$) and the following TCP parameters were set:

- window scaling was turned on, and the advertised window scaling value was 7
- SACK was enabled.

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

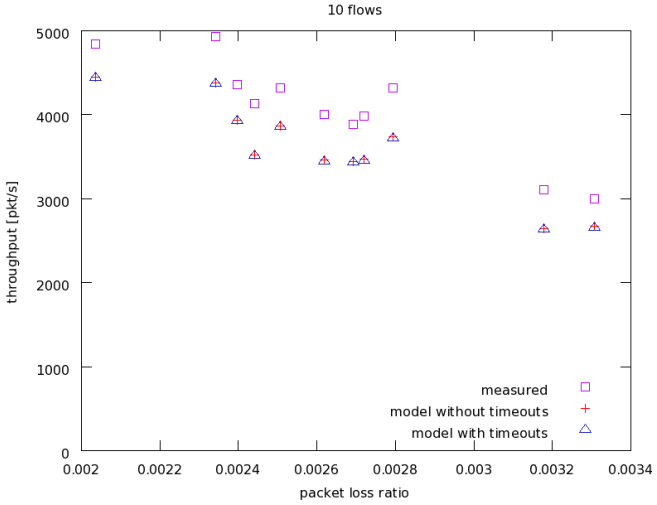
A traceroute to this host, captured during the measuring period, is listed below:

```
 1  sr01a.uibk.ac.at (138.232.65.126)  0.303 ms  0.343 ms  0.406 ms
 2  rborder.uibk.ac.at (138.232.15.222)  0.640 ms  0.896 ms  1.130 ms
 3  gigabitethernet9-1.ibk1.aco.net (193.171.19.1)  16.251 ms  16.246 ms  16.271 ms
 4  vlan353.ibk2.aco.net (193.171.15.74)  1.270 ms  1.329 ms  1.416 ms
 5  vlan352.wien21.aco.net (193.171.15.69)  11.403 ms  11.489 ms  11.542 ms
 6  aconet.rt1.vie.at.geant2.net (62.40.124.1)  11.314 ms  12.152 ms  12.034 ms
 7  so-3-0-0.rt1.mil.it.geant2.net (62.40.112.18)  24.497 ms  24.553 ms  24.548 ms
 8  so-6-3-0.rt1.gen.ch.geant2.net (62.40.112.33)  31.637 ms  31.722 ms  31.716 ms
 9  so-3-0-0.rt1.par.fr.geant2.net (62.40.112.30)  40.550 ms  40.553 ms  40.546 ms
10  renater-gw.rt1.par.fr.geant2.net (62.40.124.70)  88.814 ms  88.993 ms  88.987 ms
11  te0-0-0-0-paris2-rtr-001.noc.renater.fr (193.51.189.6)  44.126 ms  44.204 ms
    43.691 ms
12  * * *
13  telemus-2a-gi8-1-reims-rtr-021.noc.renater.fr (193.51.184.249)  44.301 ms
    44.208 ms  44.198 ms
14  * * *
15  * * *
16  * * *
17  193.55.154.4 (193.55.154.4)  62.452 ms  66.615 ms  70.953 ms
18  planetlab2.utt.fr (194.254.215.12)  71.220 ms  83.222 ms  85.360 ms
```

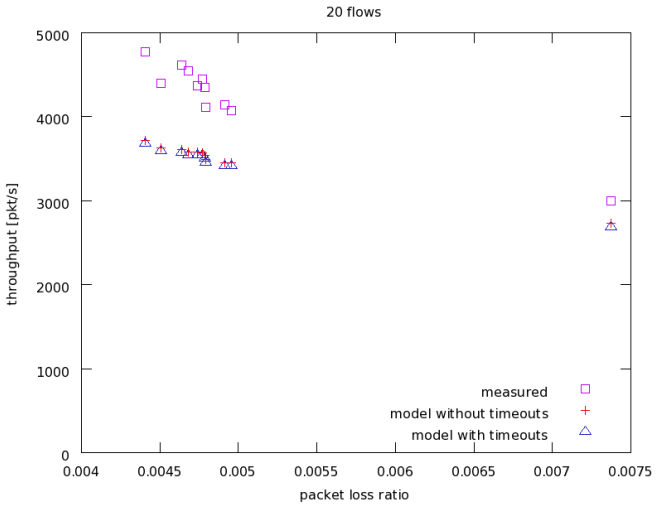
The measurements took place on 6th October 2009, from 12:00 till 18:00. Each measurement took around 320 seconds, and the last 300 seconds were used for the throughput measurement. The first 20 seconds of all transfers were neglected because we wanted to eliminate the influence of the initial slow start phase on a throughput measurement. This is similar for all following measurements. Because of a fairly short round-trip time on the path from Innsbruck to Troyes, the observed time period was long enough to capture the steady-state behavior of TCP (the average period between two loss events of the cumulative flow was 0.39% of the time period used for measurements when 10 flows were measured, and 0.26% in the case of the measurements of 20 flows). Even though it is to expect that the loss occurred in the access link (since our network card was limited to 100 Mbps), we cannot be certain. The highest throughput measured on this path was around 5000 pkt/s, which is only around 57% of 100 Mbps link. This shows that our access link could not have been the bottleneck.

As Figure 3.33 (a) shows, the model underestimated the throughput by about 600pkt/s for all measurements of 10 flows, and even more than 1000pkt/s when 20 flows were observed. By taking a closer look at the measurements, an extremely large number of consecutive losses in a loss event of the cumulative flow could have be seen. Figure 3.34 shows the number of packets lost in a loss event of the cumulative flow for each measurement[4]. To associate the measurements presented in Figure 3.33 with the corresponding number of packets lost in a loss event easier, these values are also plotted depending on measured loss rates. The values are rather large and for the measurements of 10 flows they are even greater than 10. Since we had enough information about individual flows we are able to measure the average number of individual loss events per loss event of the cumulative flow (in the following description we will refer to it as the "measured j"). We wanted to investigate how well our j approximation estimates a real value, therefore the same figure also shows these values. It can be seen that these values are much smaller, around 5. Therefore we applied the improved j to the algorithm as well. The results with the improved j are shown in Figure 3.35.

[4]This is used as an approximation of j (number of flows experiencing loss in a loss event of the cumulative flow), therefore, it is also denoted by "old j" in such figures. Since the number of flows experiencing loss in a loss event of the cumulative flow cannot be greater than the number of existing flows (n) the algorithm ensures that the old j is not greater than n. This limit is not applied for the values shown in these figures.

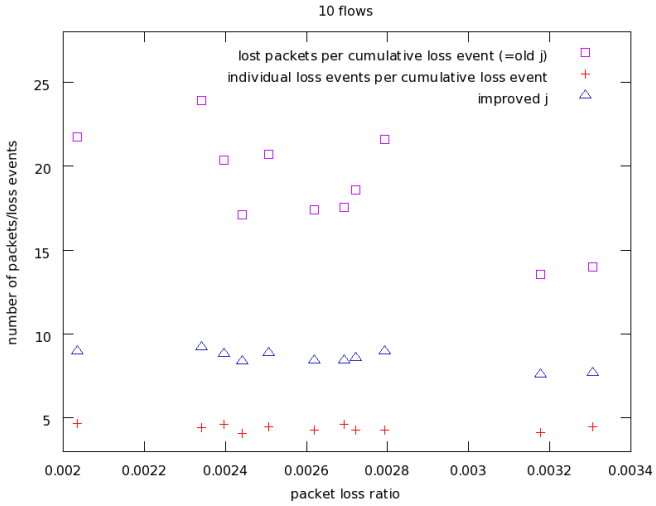


(a)

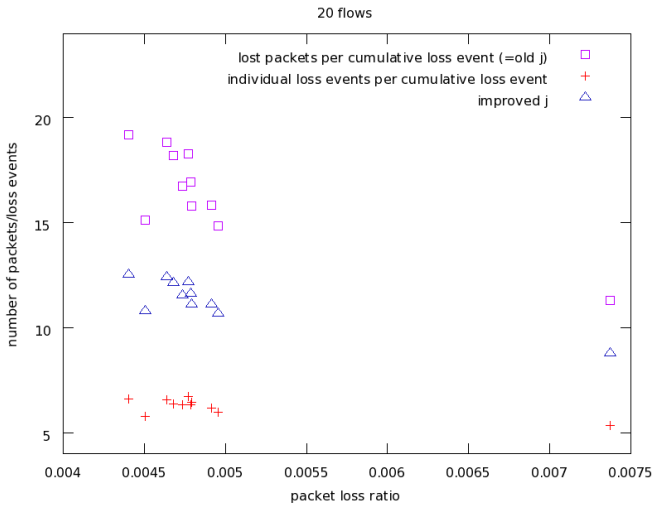


(b)

Figure 3.33: Measurements Innsbruck — France, using the equation with the loss event rate of the cumulative flow

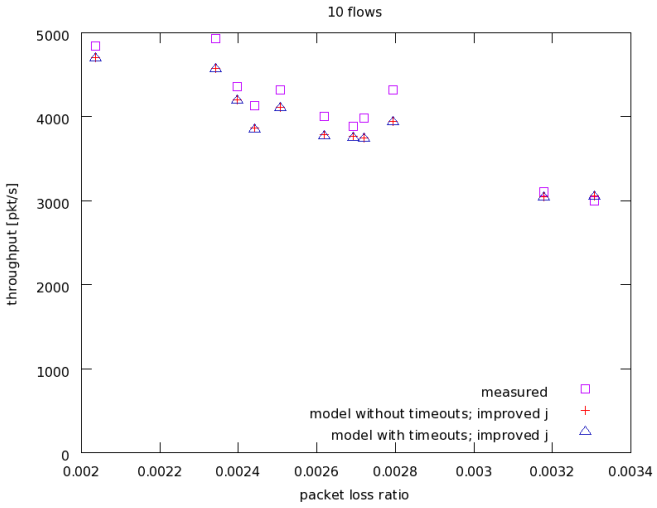


(a)

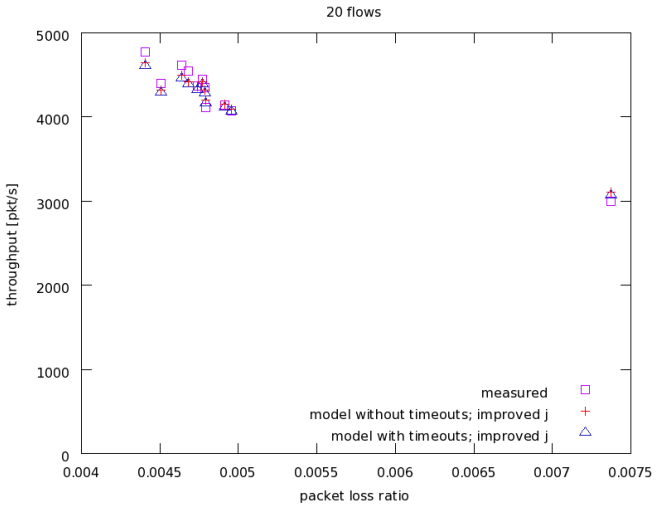


(b)

Figure 3.34: Measurements Innsbruck — France: the number of packets lost in a loss event; the number of loss event of individual flows in a loss event of the cumulative flow; the j approximation



(a)



(b)

Figure 3.35: Measurements Innsbruck — France, using the equation with the loss event rate of the cumulative flow and the improved j calculation

The results with the old j calculation (Figure 3.33) are not very satisfying, with an error of up to 15% in the case of 10 flows being observed, and even up to 22% difference in the case of the measurements of 20 flows. On the other hand, the model including the improved j deviates by less than 8% from the measured throughput of 10 flows. The throughput of 20 flows calculated with our equation deviates from the measured one just by 0.2% in some cases, and the difference never exceeds 3%. This confirms our expectation that, as the number of observed flows is increased, the influence of the error introduced by any j approximation decreases.

Innsbruck — Italy

The host at Politecnico di Milano, Milan, Italy (planetlab2.elet.polimi.it — 131.175.17.10) ran PlanetLab (kernel version 2.6.22.19$-vs$2.3.0.34.39.*planetlab*) and the following TCP parameters were set:

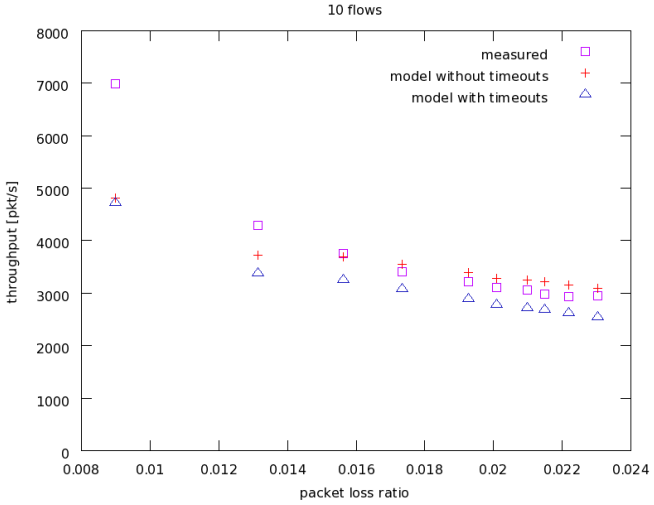
- window scaling was turned on, and the advertised window scaling value was 7
- SACK was enabled

A traceroute to this host captured during the measuring period is listed below:

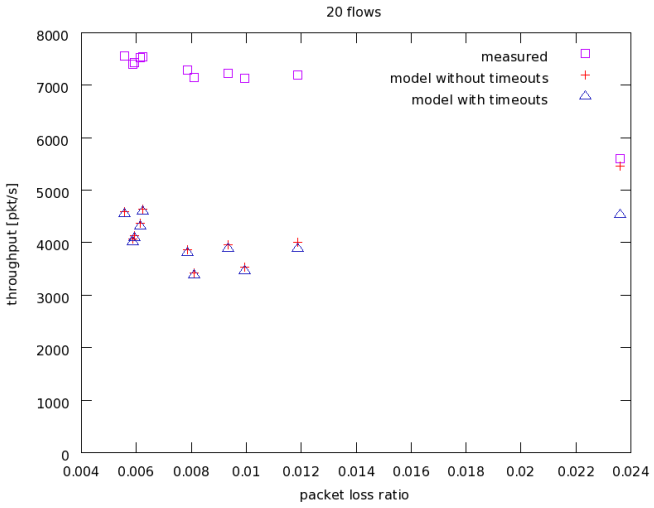
```
 1  sr01a.uibk.ac.at (138.232.65.126)  0.304 ms  0.351 ms  0.435 ms
 2  rborder.uibk.ac.at (138.232.15.222)  0.635 ms  0.956 ms  1.123 ms
 3  gigabitethernet9-1.ibk1.aco.net (193.171.19.1)  0.579 ms  0.572 ms  0.652 ms
 4  vlan353.ibk2.aco.net (193.171.15.74)  0.898 ms  0.885 ms  0.905 ms
 5  vlan352.wien21.aco.net (193.171.15.69)  11.121 ms  11.207 ms  11.415 ms
 6  aconet.rt1.vie.at.geant2.net (62.40.124.1)  11.390 ms  11.488 ms  11.314 ms
 7  so-3-0-0.rt1.mil.it.geant2.net (62.40.112.18)  23.749 ms  23.759 ms  23.752 ms
 8  garr-gw.rt1.mil.it.geant2.net (62.40.124.130)  23.722 ms  23.715 ms  24.225 ms
 9  rt1-mi1-rt-mi3.mi3.garr.net (193.206.134.138)  24.353 ms  24.444 ms  24.441 ms
10  rt-mi3-ru-polimi.mi3.garr.net (193.206.129.114)  24.518 ms  24.652 ms  24.740 ms
11  131.175.174.121 (131.175.174.121)  24.678 ms  24.702 ms  24.173 ms
12  131.175.174.226 (131.175.174.226)  24.164 ms  24.158 ms  24.189 ms
13  131.175.70.253 (131.175.70.253)  25.125 ms  25.962 ms  25.954 ms
14  planetlab2.elet.polimi.it (131.175.17.10)  25.481 ms  25.510 ms  25.511 ms
```

Each measurement had a duration of 320 seconds and the last 300 seconds were used for the throughput measurement. For the measurements of 10 flows the average time between two loss events of the cumulative flow was just 0.02% (for some measurements even less) of the duration of a measurement. Since, as we will explain later, an increase of the round-trip time was seen for the measurements of 20 flows, this percentage was 0.2%. Therefore, for some of these measurements we increased the duration of the measuring interval but the result were not changed. The loss occurred somewhere on the path and we cannot be certain if this was on our access link.

The throughput of 10 flows was measured first and the measurements were mostly taken in the period between 1st of October 2009 (18:23) and 5th of October 2009. Measuring the throughput of 20 flows followed, but, as seen in the Figure 3.36, the results are quite different. The estimated throughput, using our equation, is fairly close to the measured throughput when the measurements of 10 flows are observed (the error is around 10%). On the other hand, our equation drastically underestimates the measured throughput of 20 flows and in some cases the

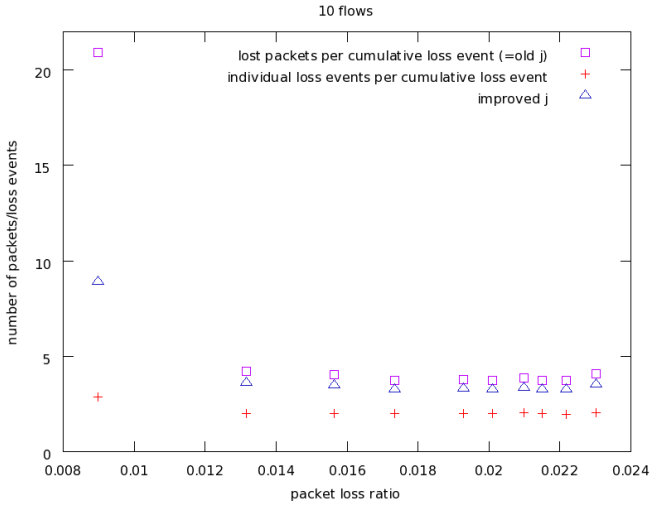


(a)

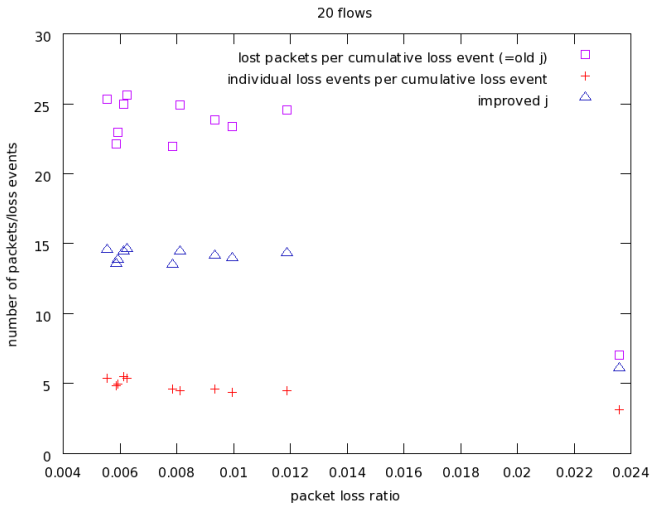


(b)

Figure 3.36: Measurements Innsbruck — Italy, using the equation with the loss event rate of the cumulative flow

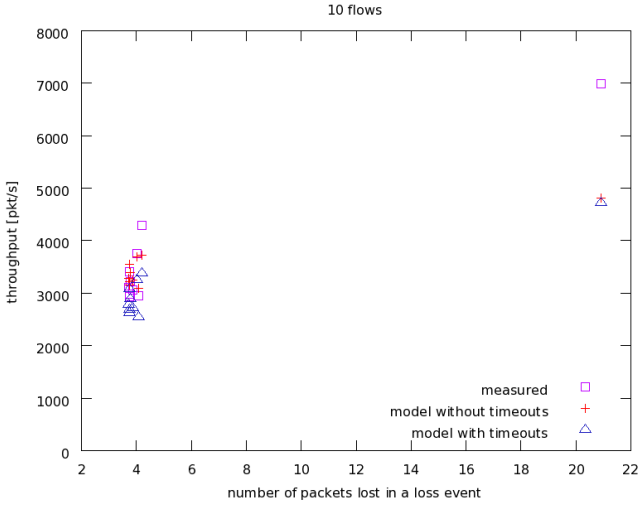


(a)

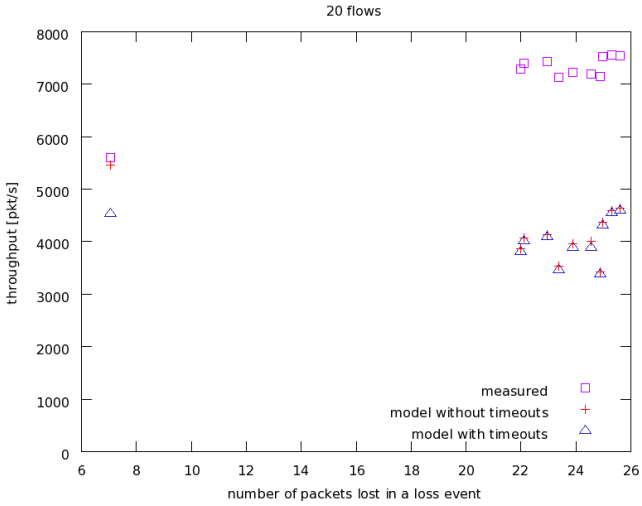


(b)

Figure 3.37: Measurements Innsbruck — Italy: the number of packets lost in a loss event of the cumulative flow; the number of loss event of the individual flows in a loss event of the cumulative flow; the j approximation

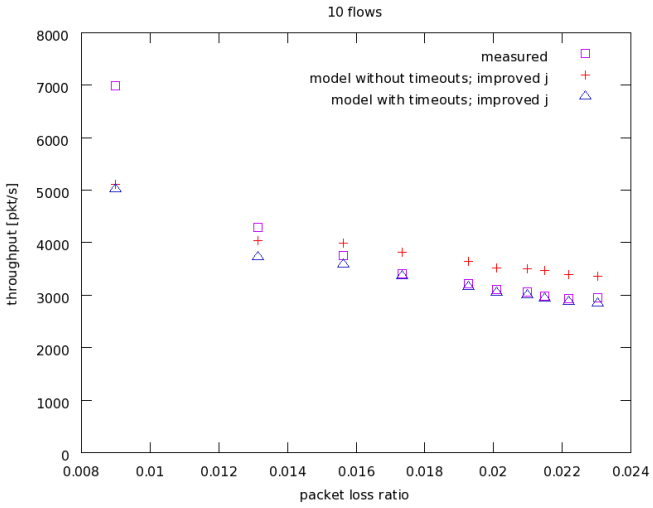


(a)

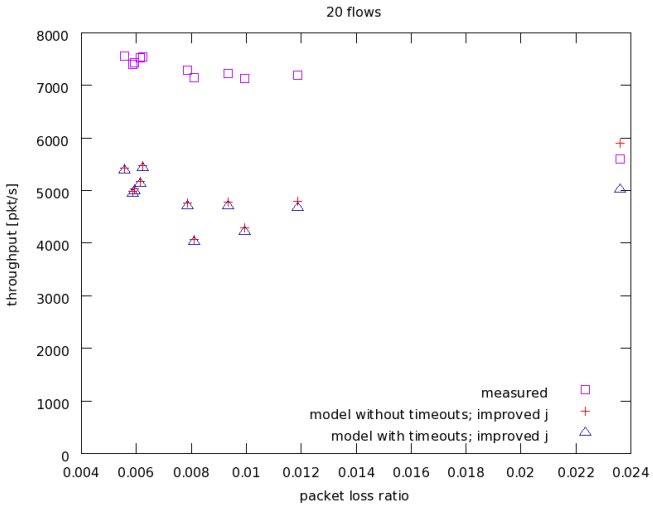


(b)

Figure 3.38: Measurements Innsbruck — Italy: the measured and calculated throughput depending on the number of packets lost in a loss event of the cumulative flow



(a)



(b)

Figure 3.39: Measurements Innsbruck — Italy, using the equation with the loss event rate of the cumulative flow and the improved j calculation

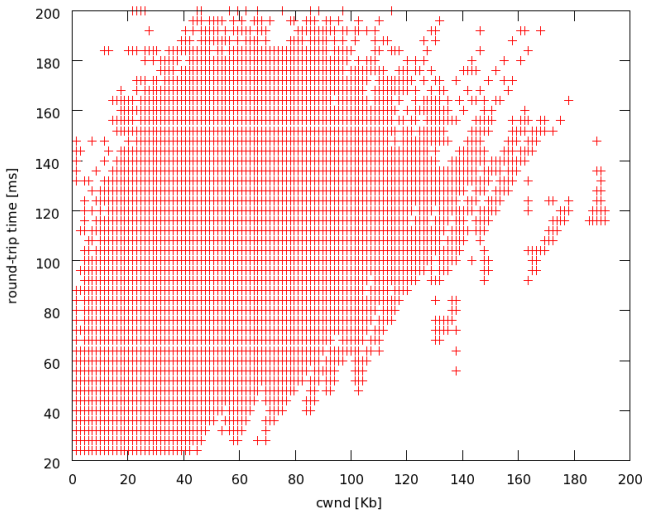


Figure 3.40: Measurements Innsbruck — Italy, the measurement taken on 9th of October 2009 (14:53): measured round-trip times vs. cwnd

equation gives only half of the measured throughput. Investigating this further, we noticed that the variation of the round-trip time was quite different in these two cases. In the second case the average round-trip time was around 80 ms (for some measurements even more than 90 ms) and during the measurements it varied between 24 ms and 190 ms (in some measurements even more). On the other hand, by observing 10 flows, such an extreme variation of the round-trip time was not seen. The average round-trip time was around 26 ms and the largest seen value was around 80 ms. We ran a test with 10 flows again immediately after a test with 20 flows to examine the possibility that this change in the round-trip time is due to increased load introduced by 10 additional flows. These tests were performed on 12th of October 2009 but, as they confirm, the increased load was not the reason for this behavior of the round-trip time. The throughput obtain in these measurements is also shown in Figure 3.36. In Figure 3.36 (a), the result with a loss rate around 0.9% corresponds to the last measurement and the equation underestimates the throughput by around 30%.

Some of the measurements of 20 flows, which had a very small loss rate (around 0.4%, see Figure 3.36 (b)), had a large number of time-outs; in some cases, even 18% of loss events were time-outs. As shown in Figure 3.37 the number of lost packets per loss event of the cumulative flow is increased by more than five times for the measurements taken after 5th of October 2009 (in Figure 3.37 (a), for measurements taken before the 5th of October this value was around 4, and for the last measurements this value was around 21). This can also be seen in Figure 3.38 which shows the measured and calculated throughput plotted against the number of lost packets in a loss event. We believed that this increased in the number of time-outs and the increased variation of the round-trip time can be explained by a change on the path that caused more burst losses to appear. As our simulations presented in Subsection 3.3.4 (see Figure 3.30)

showed, our equation can not cope well with burst loss which causes multiple packets, belonging to a small number of flows, to be lost.

Again, we applied the improved j approximation (Figure 3.39). The throughput of 10 flows was very accurately calculated by our equation. However, for the measurements taken after 5th of October 2009 (the measurements with burst loss and high round-trip time variations), the improved equation also does not give satisfactory results. The measurements between Innsbruck and the host in France (planetlab2.utt.fr) also show burst losses but, comparing Figure 3.34 and Figure 3.37, it can be seen that difference between the approximated values of j and the measured j is smaller in the case of the first measurements (Innsbruck — France). Additionally we believe that this mismatch of the results for the measurements between Innsbruck and Italy is also due to a massive increase in the round-trip time as the congestion window increases (possibly caused by an exceedingly large queue somewhere along the path), see Figure 3.40; this kind of round-trip variation is not captured by our model.

Innsbruck — Portugal

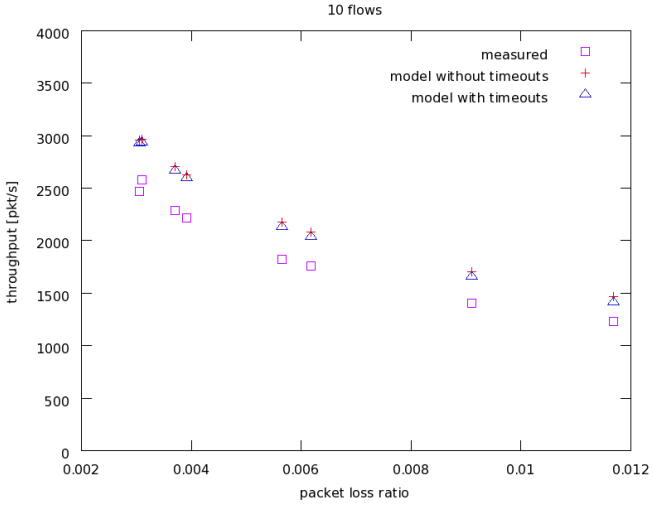
The next host was situated at Universidade do Algarve, Algarve, Portugal (the host name was planetlab1.fct.ualg.pt — 193.136.227.163). It also ran PlanetLab software (kernel version $2.6.22.19 - vs2.3.0.34.39.planetlab$) and the following TCP parameters were set:

- window scaling was turned on, and the advertised window scaling value was 7
- SACK was enabled

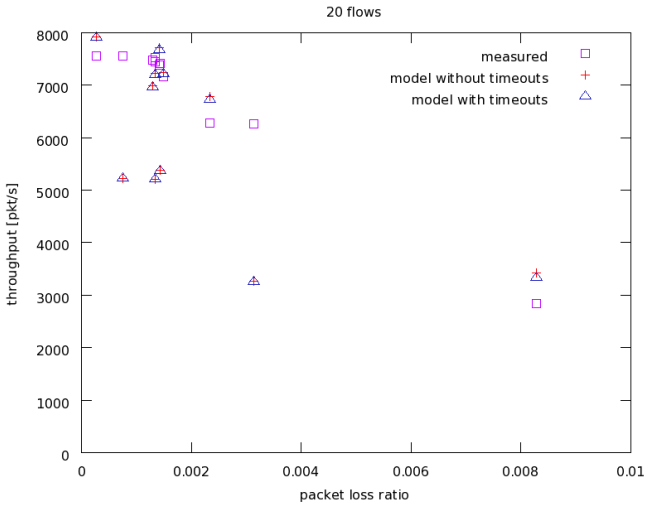
The measurements were run in the period between 2nd of October 2009 and 5th of October 2009. Each measurement had a duration of 600 seconds (the first 20 seconds are not counted) and the average time between two loss events was 0.005% of the total duration of a measurement. In this set of measurements, some transfers achieved a very high cumulative rate. It was around 8000 pkt/s, which is more than 90% of our network card capacity. Most probably, in these cases, the loss occurred in our access link. A traceroute to this host, captured during the measuring period, is listed below:

```
 1  sr01a.uibk.ac.at (138.232.65.126)  0.635 ms  0.668 ms  0.741 ms
 2  rborder.uibk.ac.at (138.232.15.222)  0.891 ms  1.211 ms  1.400 ms
 3  gigabitethernet9-1.ibk1.aco.net (193.171.19.1)  0.872 ms  0.971 ms  0.962 ms
 4  vlan353.ibk2.aco.net (193.171.15.74)  1.215 ms  1.208 ms  1.303 ms
 5  vlan352.wien21.aco.net (193.171.15.69)  11.499 ms  11.519 ms  11.826 ms
 6  aconet.rt1.vie.at.geant2.net (62.40.124.1)  11.783 ms  11.313 ms  11.620 ms
 7  so-3-0-0.rt1.mil.it.geant2.net (62.40.112.18)  24.040 ms  24.074 ms  24.107 ms
 8  so-6-3-0.rt1.gen.ch.geant2.net (62.40.112.33)  31.384 ms  31.420 ms  31.430 ms
 9  so-7-0-0.rt1.mad.es.geant2.net (62.40.112.26)  53.327 ms  53.378 ms  53.392 ms
10  fccn-gw.rt1.mad.es.geant2.net (62.40.124.98)  65.120 ms  65.124 ms  65.116 ms
11  Router3.10GE.Lisboa.fccn.pt (193.137.0.10)  65.358 ms  65.420 ms  66.945 ms
12  UALG.Faro.fccn.pt (193.137.1.2)  74.854 ms  73.292 ms  75.970 ms
13  planetlab1.fct.ualg.pt (193.136.227.163)  73.906 ms  72.889 ms  72.898 ms
```

Figure 3.41 shows results for 10 and 20 observed flows; they look quite different. In the case of 10 flows, the measurements were taken before the 5th of October 2009. The measurements of

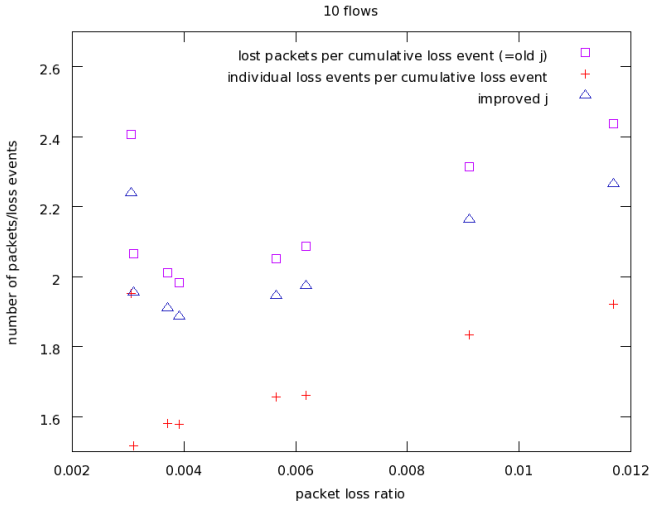


(a)

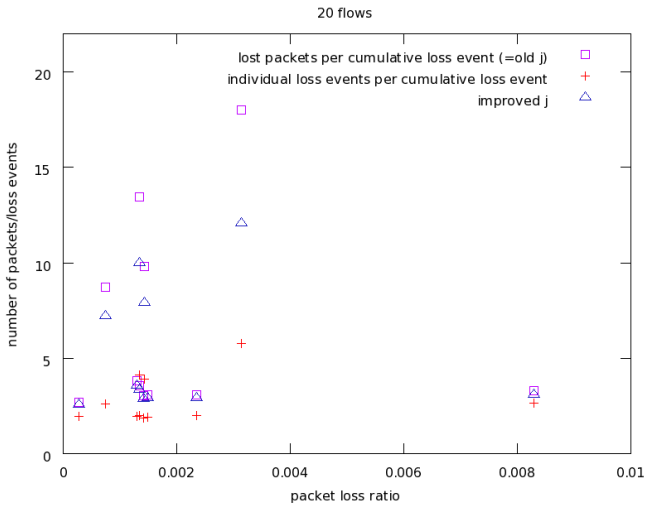


(b)

Figure 3.41: Measurements Innsbruck — Portugal, using the equation with the loss event rate of the cumulative flow

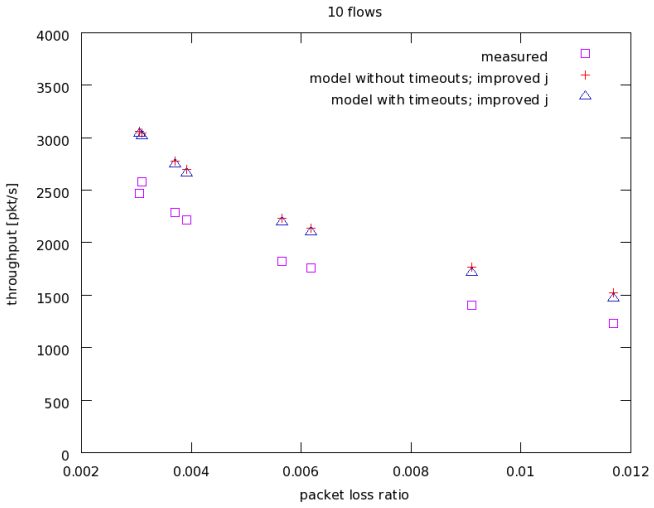


(a)

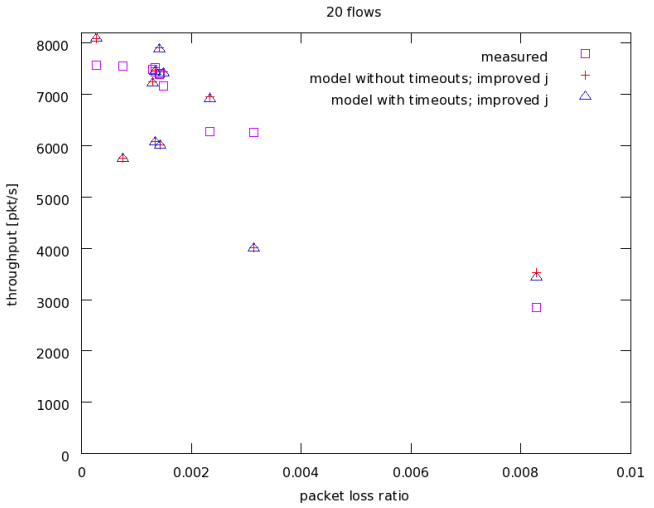


(b)

Figure 3.42: Measurements Innsbruck — Portugal: the number of packets lost in a loss event; the number of loss events of individual flows in a loss event of the cumulative flow; the j approximation

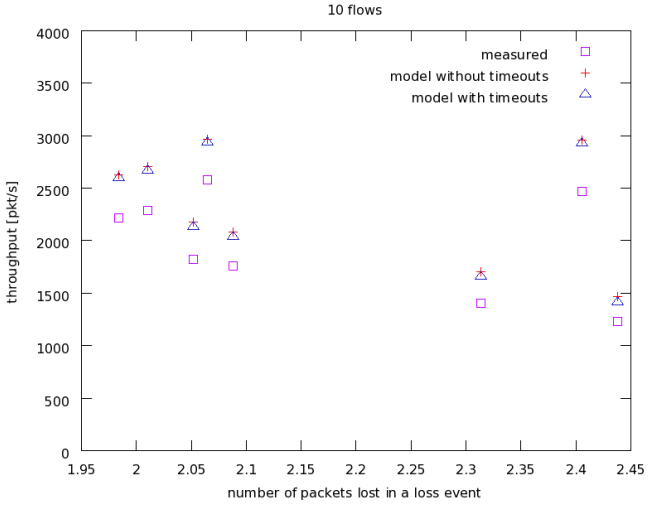


(a)

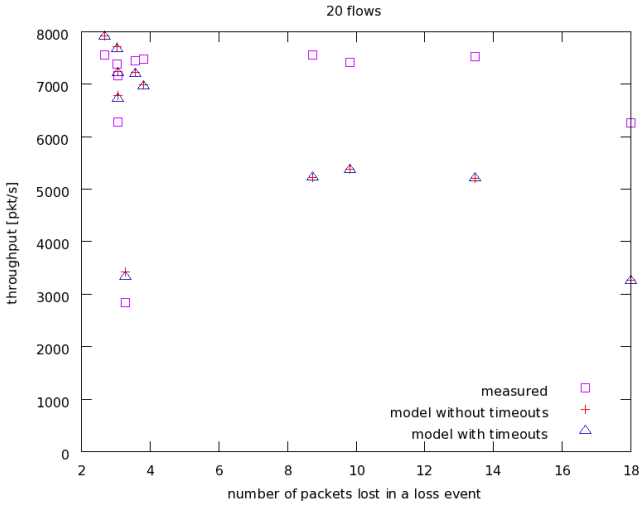


(b)

Figure 3.43: Measurements Innsbruck — Portugal, using the equation with the loss event rate of the cumulative flow and the improved j calculation



(a)



(b)

Figure 3.44: Measurements Innsbruck — Portugal: the measured and calculated throughput depending on the number of packets lost in a loss event of the cumulative flow

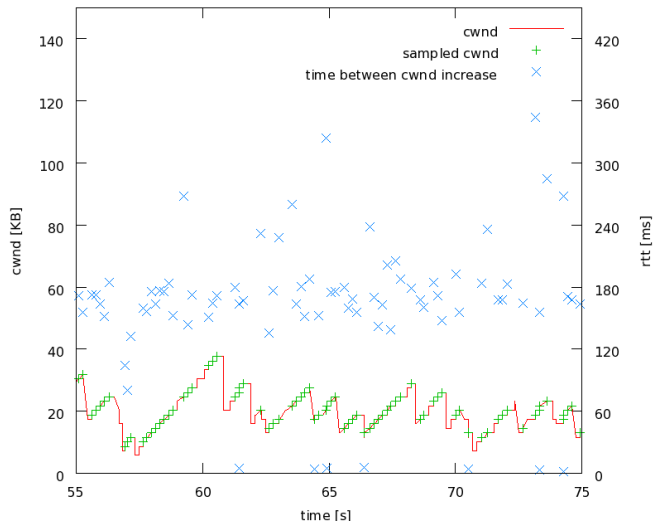


Figure 3.45: Measurements Innsbruck — Portugal: durations of sample rounds

20 flows were taken before and on 5th of October 2009 and the last measurements, taken on the 5h of October, show the same burst losses as the measurements taken between Innsbruck and Italy. Figure 3.42 shows the measured j, the number of loss events of individual flows in a loss event of the cumulative flow, and the old j approximation. In figure depicting measurements of 20 flows, values of the measured j are around 2 and the number of losses in a loss event reaches values of even 17 for some measurements (these are the same measurements for which our equation gave to some degree poor results); this can be better seen in Figure 3.44. Also the average round-trip time is increased comparing to the earlier measurements; for the earlier measurements it was around 90 ms and for the measurements from 5th October it reached 160 ms, and therefore even the use of the improved j approximation does not give satisfying results, see Figure 3.43 (b).

On the other hand, the measurements of 10 flows and a part of the measurements of 20 flows show interesting results that certainly need deeper investigations. For these measurements our equation overestimates the throughput by around 10%. The measured j values were very small, around 1.5, and the approximation that we use for our equation, did not differ much from the measured values. The use of the improved j approximation makes an even larger gap between the measurements and our equation, see Figure 3.43 (a). We notice that the round-trip time varied between 68 ms and around 200 ms and, at the same time, the average value was around 75 ms, which means that a relatively small number of packets had a longer round-trip time. Because of the way TCP operates — sending packets in bursts, it can be expected that these packets with the longer round-trip time were sent towards the end of a round and caused the time between increases of the window to be longer than the average round-trip time.

Since the Web100 kernel does not give a possibility to trace the time between window increases, we used a trace of the congestion window size and calculated the time between two increases of the window; only increases by one packet were considered. We ignored rounds that had a shorter duration because a loss was detected before their end, and also the first round after a fast recovery was omitted. Figure 3.45 shows the congestion window and sample rounds duration for one of the flows in the measurement taken on 16th of October 2009 with the start time at 11:35. The line draws the congestion window evolution, the green points present the congestion window at the beginning of a round whose duration was measured; their scale is on the left-hand side. The blue points show the duration of rounds and their scale is on the right-hand side. As it can be seen some values of the round duration are large and some are extremely small; a possible reason for this can be that the packets of one round were delayed but the packets of the consecutive round were not delayed and therefore, a quick increase of the congestion window could be seen. Taking the average of the measured round durations we obtain a higher value that the average round-trip time. Since Web100 does not provide a measurement of the round duration, we were not able to obtain precise measurements of this parameter and, for a practical application of our model, such measurements would increase the complexity of its usage. Also here, our equation shows the balance between accuracy and complexity.

Innsbruck — Uruguay

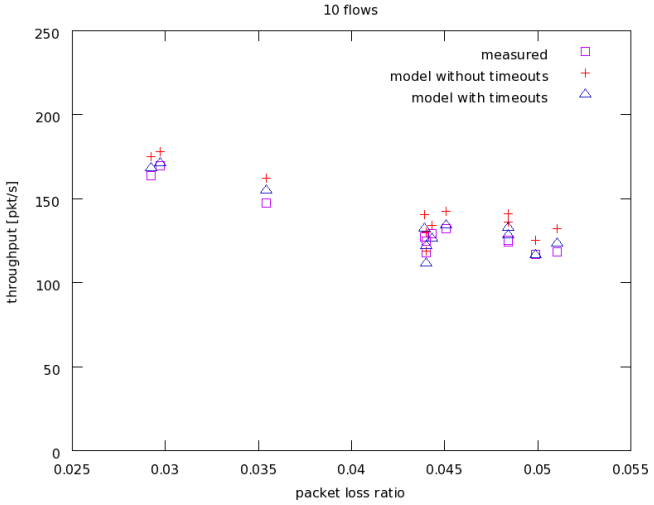
The host was situated at Facultad de Ingenieria - Universidad de la Republica, Montevideo, Uruguay (the host name: planetlab-1.fing.edu.uy — 164.73.47.242). It ran PlanetLab (kernel version 2.6.22.19 – vs2.3.0.34.39.*planetlab*) and the following TCP parameters were set:

- window scaling was turned on, and the advertised window scaling value was 7
- SACK was enabled

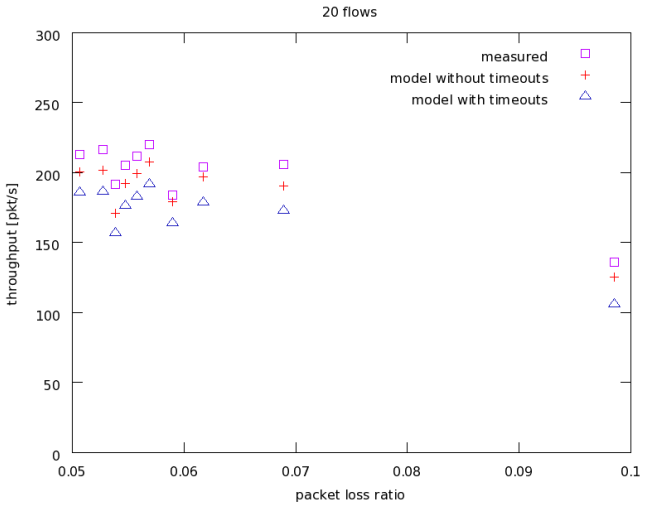
The measurements to this host were performed on 7th of October 2009; they started at 11:38 and lasted till 14:14. Even though the round-trip time on the path was quite long, 300 seconds was long enough for obtaining steady-state throughput measurements; this is because of a high loss rate seen on this path. Each connection decreased its rate more than 100 times during a measurement. The first 20 seconds are omitted here as well. On this path a high loss rate was seen but we are not able to say which links are responsible.

A traceroute to this host, taken in that period, is listed below:

```
1  sr01a.uibk.ac.at (138.232.65.126)  0.335 ms  0.389 ms  0.468 ms
2  rborder.uibk.ac.at (138.232.15.222)  0.615 ms  0.849 ms  1.069 ms
3  gigabitethernet9-1.ibk1.aco.net (193.171.19.1)  0.556 ms  0.750 ms  0.771 ms
4  vlan353.ibk2.aco.net (193.171.15.74)  0.909 ms  0.896 ms  0.924 ms
5  vlan352.wien21.aco.net (193.171.15.69)  11.345 ms  11.332 ms  11.411 ms
6  aconet.rt1.vie.at.geant2.net (62.40.124.1)  11.233 ms  11.159 ms  11.711 ms
7  so-3-0-0.rt1.mil.it.geant2.net (62.40.112.18)  24.382 ms  24.375 ms  24.443 ms
```

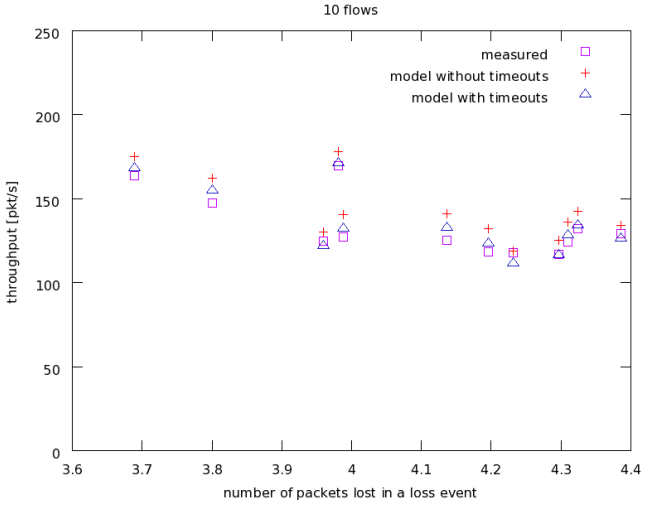


(a)

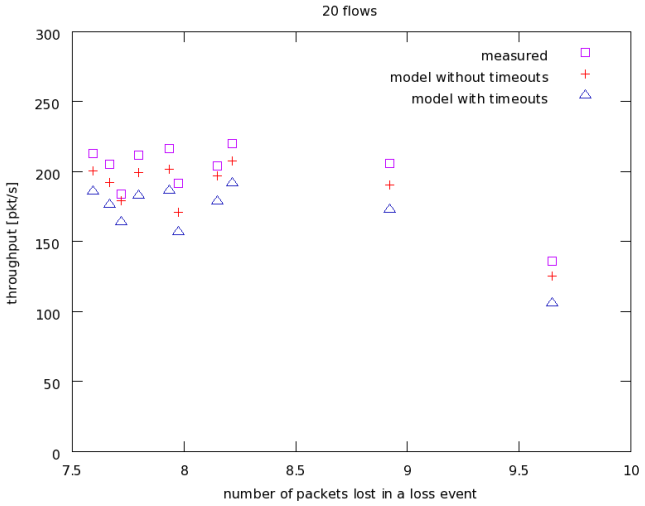


(b)

Figure 3.46: Measurements Innsbruck — Uruguay, using the equation with the loss event rate of the cumulative flow

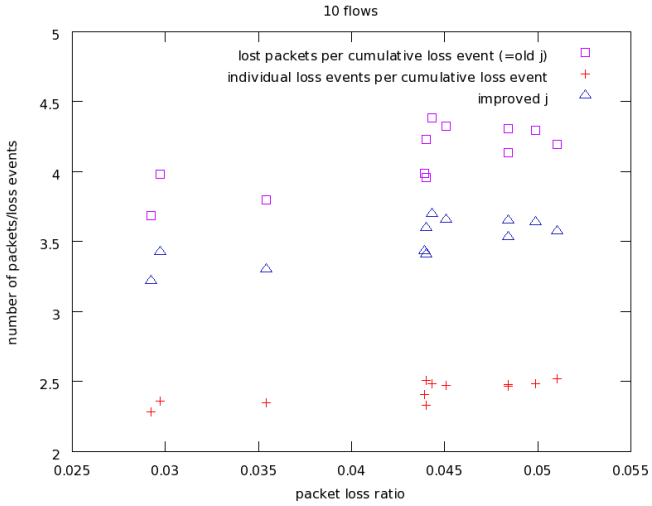


(a)

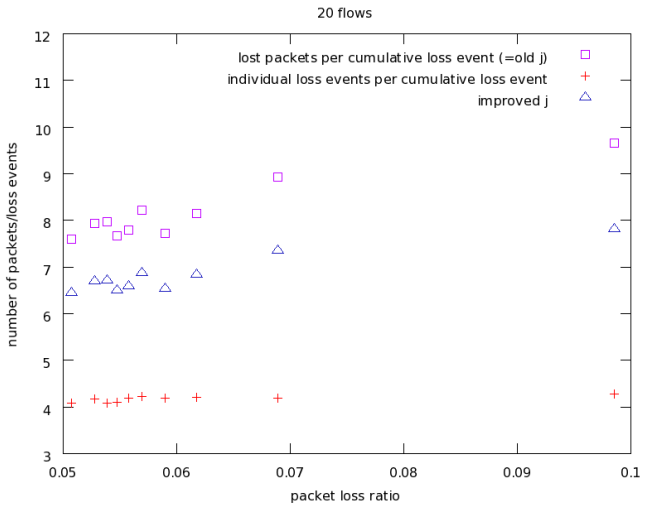


(b)

Figure 3.47: Measurements Innsbruck — Uruguay: measured and calculated throughput depending on the number of packets lost in a loss event of the cumulative flow



(a)



(b)

Figure 3.48: Measurements Innsbruck — Uruguay: the number of packets lost in a loss event; the number of loss events of individual flows in a loss event of the cumulative flow; the j approximation

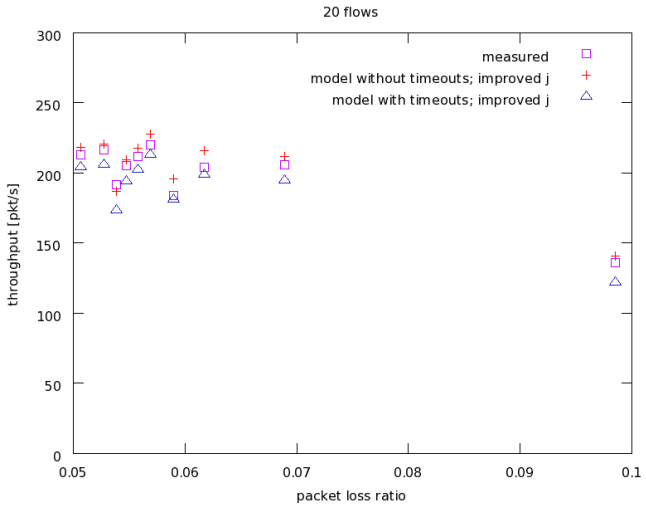


Figure 3.49: Measurements Innsbruck — Uruguay, using the equation with the loss event rate of the cumulative flow and the improved j calculation

```
 8  so-6-3-0.rt1.gen.ch.geant2.net (62.40.112.33)  31.537 ms  31.530 ms  31.544 ms
 9  so-7-0-0.rt1.mad.es.geant2.net (62.40.112.26)  53.675 ms  53.678 ms  53.672 ms
10  clara-gw.rt1.mad.es.geant2.net (62.40.124.62)  167.806 ms  167.550 ms  167.542 ms
11  200.0.204.41 (200.0.204.41)  275.347 ms  275.343 ms  275.331 ms
12  saopaulo-buenosaires.core.redclara.net (200.0.204.37)  321.148 ms  321.111 ms
    321.101 ms
13  santiago-buenosaires.core.redclara.net (200.0.204.30)  318.870 ms  318.584 ms
    318.575 ms
14  rau-uy-bsas.peer.redclara.net (200.0.204.154)  364.185 ms  364.321 ms  364.388 ms
15  r3.rau.edu.uy (164.73.128.129)  365.221 ms  366.055 ms  365.893 ms
16  eth-fing.rau.edu.uy (164.73.253.34)  344.295 ms  342.895 ms  342.869 ms
17  164.73.32.126 (164.73.32.126)  342.484 ms  342.047 ms  342.139 ms
18  planetlab-1.fing.edu.uy (164.73.47.242)  365.571 ms  365.729 ms  365.723 ms
```

Figure 3.46 shows measurement results for the host situated in Uruguay. Here our equation approximated TCP's real behavior quite well. The measurements for 10 flows are slightly overestimated by our equation and the difference is less than 4%. On the other hand, the throughput of 20 flows is slightly underestimated.

As in the previously presented measurements, an error introduced by the use of the old j approximation and the difference between the average round-trip time and the average duration of a round can cause this mismatch. Figure 3.47 shows the measured throughput and its estimate achieved using our equation depending on the old j approximation. In measurements of 10 flows j varied less strongly and the difference between measured throughput and its estimate is almost the same in all cases. For small j values the equation slightly overestimates the throughput (this difference is still less than 4%) and, as discussed in previous measurements, this can be due to the difference between the average round-trip time of all packets and the actual average duration of a round. Large j values magnify the error from the j calculation and, at

the same time, annul the error from the approximation of the average round duration. When 20 flows were observed the error from the j approximation was larger (see Figure 3.48 — the difference between measured and approximated j values) and for all measurements our equation underestimates the throughput with the gap slightly growing as j increases (and thereby the error from the j approximation grows).

As it can be seen in Figure 3.48 (a), even the old j calculation does not overestimate the measured j by more than 60% (this looks like a large number but our equation can estimate the throughput well even in these cases, especially when the number of flows being observed is high). The same figure shows results for 20 flows; the old j approximation gives a value that is more than two times larger than the measured j and therefore, in this case, the improved j calculations gives better results, see Figure 3.49. With the improved j calculation the difference between measured throughput and its estimate, in some cases, is only between 1% and 2% and it never exceeds 10%.

Innsbruck — Korea

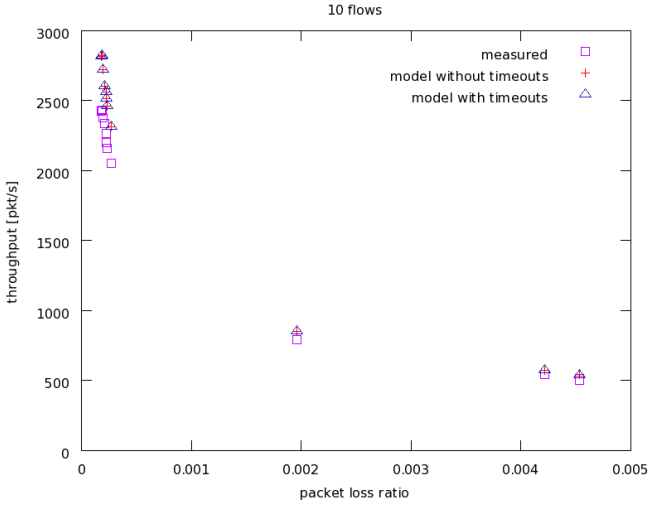
The host at Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea (the host name: csplanetlab2.kaist.ac.kr — 143.248.139.55) ran PlanetLab (kernel version $2.6.22.19 - vs2.3.0.34.39.planetlab$) and the following TCP parameters were set:

- window scaling was turned on, and the advertised window scaling value was 7
- SACK was enabled

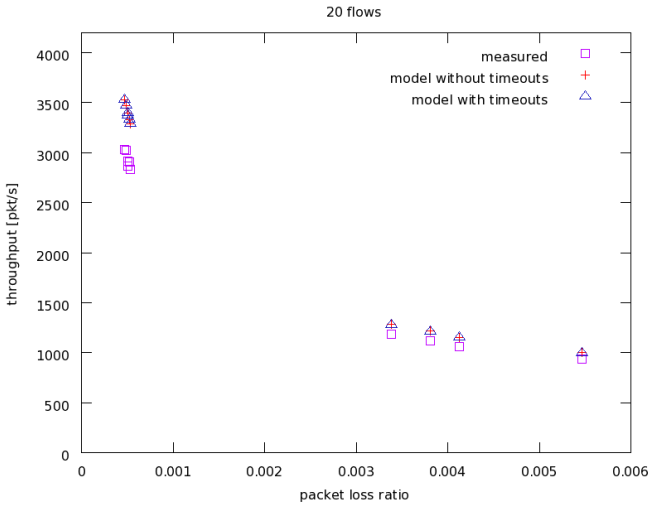
These measurements were taken starting from 7th of October 2009 and they were run till 9th of October 2009. The duration of each transfer was 645 seconds and the last 600 seconds are used for the throughput measurements. Each flow decreased its congestion window more than 100 times during a measurement. Also here, it is not certain which part of the path between Innsbruck and Korea was the bottleneck.

A traceroute to this host, taken during the measuring period, is listed below:

```
 1  sr01a.uibk.ac.at (138.232.65.126)  0.453 ms  0.513 ms  0.595 ms
 2  rborder.uibk.ac.at (138.232.15.222)  0.589 ms  0.841 ms  1.122 ms
 3  gigabitethernet9-1.ibk1.aco.net (193.171.19.1)  0.531 ms  0.677 ms  0.664 ms
 4  vlan353.ibk2.aco.net (193.171.15.74)  0.823 ms  1.023 ms  1.010 ms
 5  vlan352.wien21.aco.net (193.171.15.69)  11.094 ms  11.177 ms  11.208 ms
 6  212.73.203.17 (212.73.203.17)  128.893 ms  86.294 ms  84.200 ms
 7  ae-9-9.ebr1.Frankfurt1.Level3.net (4.69.141.158)  22.766 ms  22.831 ms  22.917 ms
 8  ae-61-61.csw1.Frankfurt1.Level3.net (4.69.140.2)  33.946 ms
    ae-71-71.csw2.Frankfurt1.Level3.net (4.69.140.6)  31.583 ms  31.556 ms
 9  ae-82-82.ebr2.Frankfurt1.Level3.net (4.69.140.25)  22.952 ms  22.986 ms
    ae-92-92.ebr2.Frankfurt1.Level3.net (4.69.140.29)  22.927 ms
10  ae-41-41.ebr2.Washington1.Level3.net (4.69.137.50)  114.360 ms
    ae-44-44.ebr2.Washington1.Level3.net (4.69.137.62)  116.111 ms
    ae-43-43.ebr2.Washington1.Level3.net (4.69.137.58)  114.806 ms
11  ae-62-62.csw1.Washington1.Level3.net (4.69.134.146)  116.345 ms
    ae-92-92.csw4.Washington1.Level3.net (4.69.134.158)  113.002 ms
    ae-72-72.csw2.Washington1.Level3.net (4.69.134.150)  113.863 ms
12  ae-64-64.ebr4.Washington1.Level3.net (4.69.134.177)  127.849 ms
    ae-74-74.ebr4.Washington1.Level3.net (4.69.134.181)  129.045 ms
```

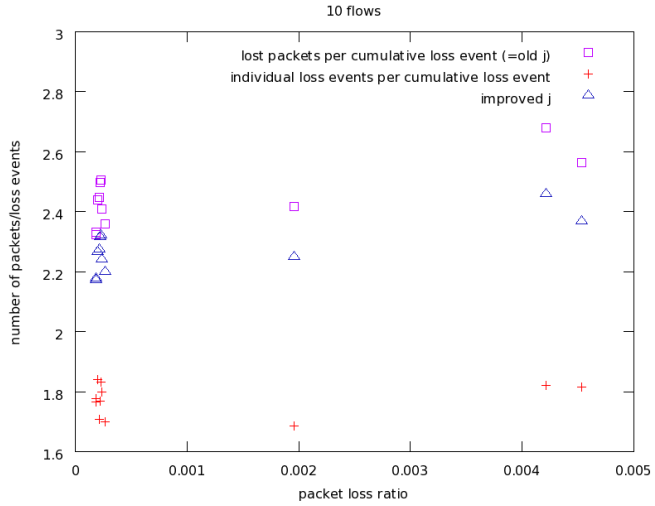


(a)

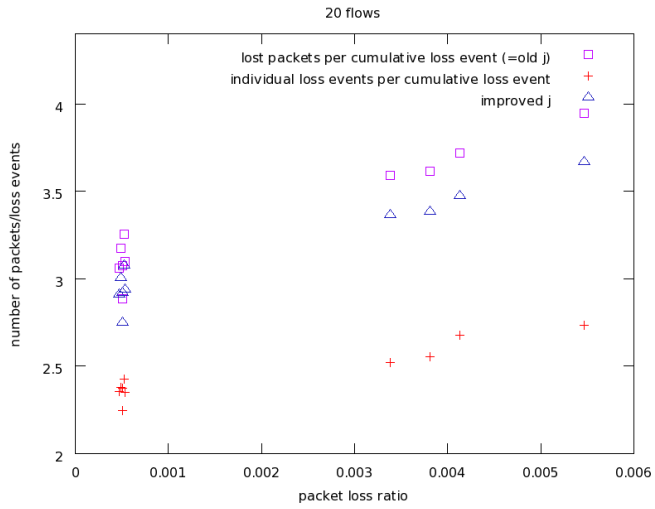


(b)

Figure 3.50: Measurements Innsbruck — Korea, using the equation with the loss event rate of the cumulative flow

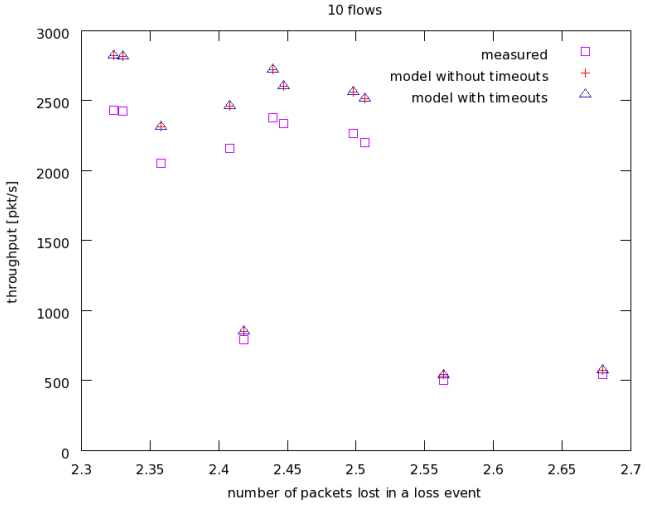


(a)

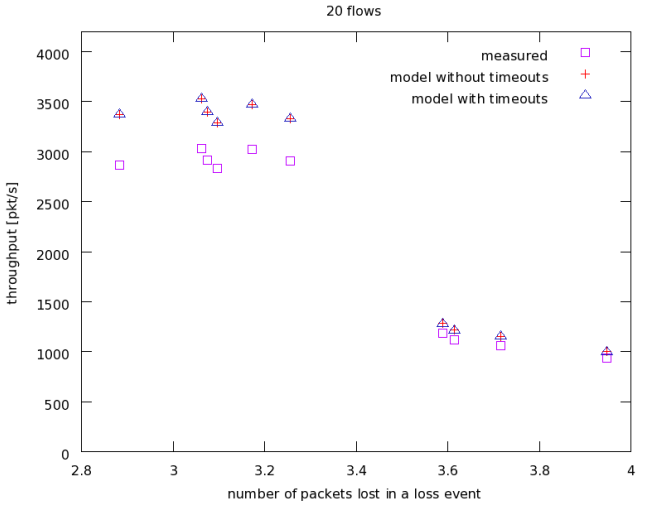


(b)

Figure 3.51: Measurements Innsbruck — Korea: the number of packets lost in a loss event; the number of loss events of individual flows in a loss event of the cumulative flow; the j approximation



(a)



(b)

Figure 3.52: Measurements Innsbruck — Korea: the measured and calculated throughput depending on the number of packets lost in a loss event of the cumulative flow

```
    ae-94-94.ebr4.Washington1.Level3.net (4.69.134.189)  126.240 ms
13  ae-4.ebr3.LosAngeles1.Level3.net (4.69.132.81)  185.386 ms  179.905 ms
    179.896 ms
14  ae-43-90.car3.LosAngeles1.Level3.net (4.69.144.197)  180.188 ms
    ae-13-60.car3.LosAngeles1.Level3.net (4.69.144.5)  178.613 ms
    ae-23-70.car3.LosAngeles1.Level3.net (4.69.144.69)  178.488 ms
15  HANARO.car3.LosAngeles1.Level3.net (4.71.32.58)  177.674 ms  187.791 ms
    176.393 ms
16  58.229.14.197 (58.229.14.197)  320.899 ms  320.123 ms  330.692 ms
17  211.108.90.2 (211.108.90.2)  331.011 ms  331.104 ms  330.998 ms
18  118.221.5.50 (118.221.5.50)  448.614 ms  448.019 ms  447.981 ms
19  218.236.231.86 (218.236.231.86)  311.243 ms  311.976 ms  312.684 ms
20  143.248.117.17 (143.248.117.17)  312.057 ms  323.264 ms  310.113 ms
21  143.248.117.115 (143.248.117.115)  309.887 ms  330.109 ms  311.384 ms
22  csplanetlab2.kaist.ac.kr (143.248.139.55)  310.852 ms  326.007 ms  325.618 ms
```

Figure 3.50 (also Figure 3.52) shows a comparison between the measured throughput and our equation with the old j approximation. While it matches the measurements quite well, our equation slightly overestimates the throughput in all samples. Figure 3.51 shows the old j and the measured j (the number of loss events of individual flow in a loss event of the cumulative flow). As can be seen in this figure, both values, for all measurements, are very small and the error introduced by our approximation is also small (it does not exceeds 45%). As in the previously presented measurements, the difference between our model and the measured throughput is probably caused by the mismatch between the average round-trip time value and the average duration of a round. Use of the improved j calculation would only increase the error.

Conclusion

In most of the measurements our equation estimated accurately the throughput of flows as seen in Figures 3.31 (a), 3.32 (a), 3.35, 3.39 (a), 3.43, 3.46 (a) and 3.49. In some cases, the old j approximation showed greater accuracy and, in other, the improved j had to be applied. For a small number of measurements both solutions differed significantly from the measured throughput.

For all presented measurements, the difference between the measured throughput and our equation can be explained by two factors: an error introduced by the used j approximation and an error introduced by an imperfect measurement of the average duration of a round. The difference between the real number of flows experiencing loss events in a loss event of the cumulative flow and our approximation, calculated as the average number of packets lost in a loss event, ranges from 40% (measurements between Innsbruck and Portugal and between Innsbruck and Korea) to even 4 times larger in cases when we suspect, a DropTail queue with a larger buffer was present on the path (measurement between Innsbruck and Italy). As the difference grows our equation gives a lower throughput. For a smaller difference, it overestimates throughput; we believe that this is due to a error introduced by the approximation of the average round duration, with our estimate being shorter. When the j approximation overestimates the measured number of flows hit in a loss event of the cumulative flow by around 60% these two errors annul each other.

We applied two different j approximations. Whether the old j or the new j should be used, we believe it depends on the average number of packets lost in a loss event of the cumulative flow

(this is equal to p_r/p_e — packet loss rate divided by loss event rate of the cumulative flow). Looking at the obtain measurements, we came to the following conclusion:

- in case p_r/p_e is larger than 40% of the number of observed flows the new j approximation should be used. This is the case for the measurements of 10 flows: Innsbruck-France (Figure 3.35 (a)) and Innsbruck-Italy (Figure 3.39 (a)). For 20 flows this border is around 30% as it can be seen in the case of the measurements: Innsbruck-France (Figure 3.35 (b)), partially Innsbruck-Portugal (Figure 3.43 (b)) and Innsbruck-Uruguay (Figure 3.49 (b)).

- in case p_r/p_e is smaller than 40% (or 30% for 20 flows) of the number of flows the old approximation should be used. This can be seen for the measurements: Innsbruck-Korea (Figure 3.50), Innsbruck-Uruguay (Figure 3.49), Innsbruck-Portugal (Figure 3.43 (a) and partially (b))

3.4 Comparison with the equation from [103]

Due to the simplicity of our model, it is tempting to believe that a similar result could also be obtained by simply multiplying the original equation from [103] with the number of flows n. We compared both versions of the equation. On the one hand, we have the equation from Section 3.2.2 that uses the loss event rate on a per-flow basis as the only information. On the other hand, the equation from Section 3.3.2 uses two pieces of information: real loss and the loss event rate of the cumulative flow. We made a comparison of $n \times$ the equation from [103] with both equations, applying a corresponding loss event rate measure. We compared $n \times$ the equation from [103] with the equation from Section 3.2.2, feeding in the loss event rate measured on a per-flow basis, and we compared $n \times$ the equation from [103] with the equation from Section 3.3.2, feeding in the loss event rate of the cumulative flow. Having these two completely different ways of measuring loss, we expect that $n \times$ the equation from [103] calculated with these two loss measures will yield two completely different results.

For both comparisons we used the same simulation as in Section 3.2.4 (the same network setup as in the case of Figure 3.7 (a): 30 ms bottleneck delay, RED queue, random loss on the bottleneck link). As expected, $n \times$ the equation from [103] works reasonably well with the measure of the loss event rate of any flow. We observed that this calculation has an error that grows with increasing loss; this is shown in Figure 3.53. Our model gives a good estimate of the throughput even when the loss rate is as high as 10%.

The equation from Section 3.3.2 uses the loss event rate of the cumulative flow. As we assumed, our equation gives significantly better results than the equation from [103] multiplied by the number of flows in this case (Figure 3.54).

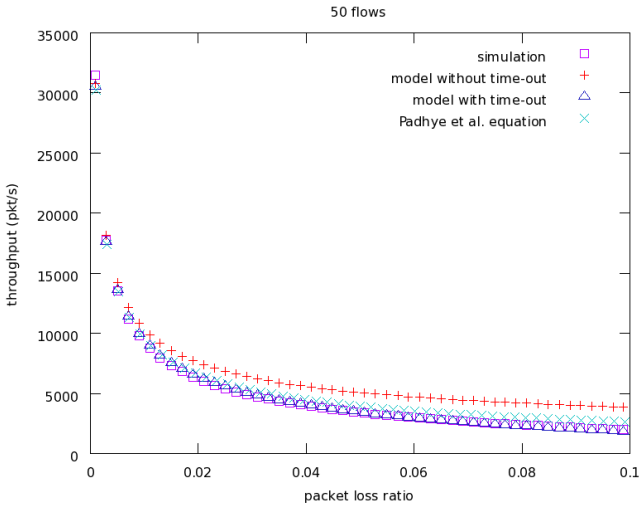


Figure 3.53: Comparison: the equation from Section 3.2.2 with $n \times$ the equation from [103]

One special case worth considering is $n = 1$: here, our equation is used for only one flow just like the original equation from [103]. With the first (simpler) model, the two equations only differ in that the Padhye et al. model assumes all packets from a window to be lost when any packet is lost, whereas we do not make this assumption for the congestion avoidance case and assume significantly clustered losses (causing time-outs) to be a rare special case. As we have argued at the outset of this chapter, our assumption seems to be more realistic in today's Internet, and we found that, indeed, our model matches our real-life measurements even for the $n = 1$ case more precisely.

3.5 Conclusion and applicability

Despite numerous examples of the TCP models presented throughout the literature, to the best of our knowledge the equations presented here are the first models of parallel TCP flows that show great accuracy and at the same time are quite easy to use. Disparately to the other models of multiple flows (e.g., [98, 99, 86, 6, 17, 17, 15, 16, 10, 30, 57, 56, 14]), our models do not need any additional information about the precise network topology and existing traffic in a network, which greatly expands the area of possible applicability. The models give the throughput of parallel TCP flows depending on the network conditions that are characterized only by the packet loss rate and the round-trip time.

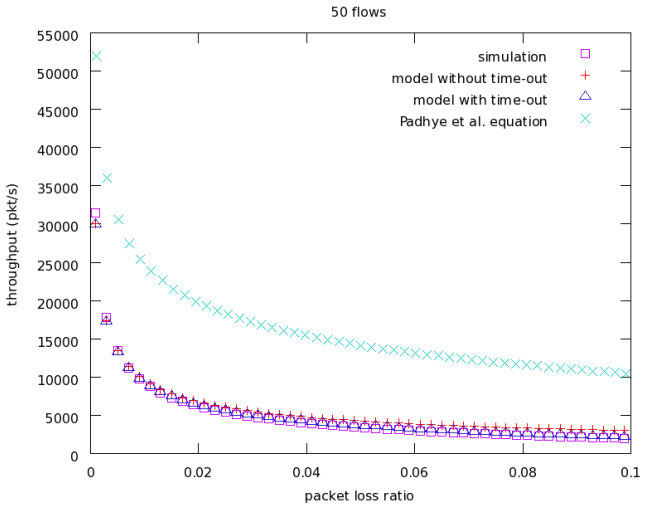


Figure 3.54: Comparison: the equation from Section 3.3.2 with $n \times$ the equation from [103]

Among the models of a single TCP flow the throughput equation from [103] is the most widely used, however, the TCP model described in section 3.2 gives a more accurate estimate of the throughput of parallel TCP flows than the equation for the single flow [103] multiplied by n (see section 3.4). The loss event rate measured on a per-flow basis is more demanding than observing all flows as a cumulative flow and the second equation, proposed in section 3.3, takes this slightly different loss assumption, which makes a comparison with $n \times$ the equation for [103] rather unfair, as discussed in more detail in section 3.4.

Because of the simplicity of the equations we presume that they can have a wide applicability. To give a glance on the possible topics, some straightforward ideas will be discussed below.

Accurately predicting the throughput of TCP connections, and thereby the latency of transfers, is of benefit for many application areas (eg., file transfers in a Grid). And further, the prediction of the throughput of parallel TCP flows becomes more interesting since the number of applications using multiple flows grows (GridFTP, bbcp, peer-to-peer applications etc.). The equation from [103] is used in [63] for predicting the throughput of a TCP connection. The authors of [63] propose two methods for predicting the throughput: formula-based and history-based. Here we are interested in the formula-based prediction, since a similar application of the equation presented in this thesis can be proposed. Prediction using the formula-based method as explained in [63] has certain drawbacks, and it can be assumed that the same problems will also appear in the case of a rate prediction for multiple flows using our equations. As stated in [63], a predicted packet loss rate and the round-trip time prior to a transfer takes place will yield a good estimate only if the newly added flow does not introduce a significant load to the network. The multiple flows have a higher aggression and therefore a greater probability to

change the network state (i.e., the packet loss rate and the round-trip time), and hence the accuracy of this prediction is questionable. The authors in [89] propose a way of estimating the change of the network conditions depending on the number of added flows, and a similar technique could be applied for a formula-based throughput prediction using our equation. Here two equations are presented with different loss measure assumptions; therefore a more profound study of these equations in this context is necessary. The chosen loss measuring technique will have a great influence on studies of this kind. For example, the authors in [63] propose measuring the loss rate and the round-trip time using "pings". Whether this measure better estimates the per-flow loss measure (used in the equation presented in section 3.2) or the loss rate of the cumulative flow (used for the equation in section 3.3) is not certain.

MulTFRC, the second major achievement of this thesis is a rate-based protocol. A single MulTFRC flow behaves like n standard TCP flows. It uses the equation proposed in section 3.3 to calculate the rate of the multiple flows. This protocol will be presented in the following chapter.

Chapter 4

MulTFRC

MulTFRC is a rate based protocol with the feature of providing the possibility for a single flow to behave like a number of standard TCP flows, thereby allowing a user to change the "aggression" of a connection; in other words, it offers a weighted congestion control. With "TCP-friendliness" being defined as the behavior of a protocol that is sharing links similar to conforming TCP flows, this protocol is an "n-TCP-friendly" protocol with n being any positive real number, even including values between zero and one.

The protocol is based on the TFRC protocol proposed in [50, 51]. The TFRC protocol provides a TCP-friendly transport service for multimedia traffic and is also a part of Datagram Congestion Control Protocol (DCCP) CCID 3 [55, 54]; DCCP is a transport protocol that offers congestion control for unreliable traffic. In TFRC, the rate of a transfer is calculated using the equation from [103]. It needs measurements of the round-trip time and the loss event rate to estimate the sending rate.

MulTFRC takes the same principle, but with the idea of offering a weighted congestion control. The equations derived in the previous chapter give the throughput of parallel TCP flows depending on the round-trip time and the loss event rate, similar to the equation from [103]. The equation using the loss measure of the cumulative flow (equation 3.55, see also algorithm 2 on page 72) will be used here.

Some aspects of the equation from Section 3.3.2 need to be refined to better suit the MulTFRC protocol design; this is analyzed and specified in the next section. Since the MulTFRC protocol offers a slightly different service and also uses a different equation, TFRC needs to be extended too (this is also part of the next section). The protocol design is followed by an extensive simulation study of the outcome which shows that the possible values of the aggression parameter n are hardly limited. Further, MulTFRC is compared with other proposed protocols that

implement a weighted congestion control (Section 4.2). Finally, a real implementation of this protocol is developed. Some details about this implementation as well as a comparison of the MulTFRC protocol and the Linux implementation of TCP are presented in Section 4.3. At the end, a short discussion of possible applications of this protocol is given.

4.1 Design

As an extension of TFRC, for multiple parallel TCP flows, MulTFRC requires an equation which provides an appropriate throughput calculation. Since MulTFRC is in fact only one flow that just behaves like n flows, and since n should be a positive real number, it is not possible to reasonably distinguish which of these n "virtual" flows a packet belongs to, and it is therefore not possible to measure the loss event rate of these (non-existent) individual flows. Hence we used the equation derived in Subsection 3.3.2 that offers us exactly what is necessary. The equation needs network characteristics as input parameters. By the network characteristics, the round-trip time, the retransmission timeout value (i.e., the initial period of time (in a time-out phase) after which the sender retransmits unacknowledged packets), the loss event rate and the number of packets lost in a loss event are meant. There are some parameters that are set depending on the desired use of the protocol, like b and n. Parameter b is the number of packets acknowledged by an ACK, and it depends on the TCP variant that we want to emulate, e.g., whether the delayed ACK and/or the Appropriate Byte Counting TCP mechanisms are used. The last parameter is the aggression parameter — n.

The sending rate of a MulTFRC flow is calculated using algorithm 2 on page 72. The output of the algorithm is the rate in $[packet/s]$, and to obtain the rate in $[bit/s]$ the result is multiplied by the packet size (s).

Further, the input parameters for this algorithm need to be measured. Some of these parameters already exist in the TFRC specification and the methods for estimating them are present, but some others are specific just for MulTFRC; more details are given in the following subsection.

4.1.1 Measuring congestion

Many input parameters for the rate calculation of MulTFRC are measured in the same way as proposed for TFRC, e.g., RTT, t_RTO and the loss event rate. For completeness and easier understanding of the MulTFRC protocol, some of these measurement approaches will be repeated here.

For RTT and t_RTO calculations (as in TFRC), the standard algorithm implemented in the TCP protocol is used. RTT is the exponential weighted moving average of round-trip time samples, and t_RTO is calculated using the algorithm specified in [21].

The loss event rate is measured for the cumulative flow. It is the same as the measurement of the loss event rate used in TFRC. As defined in Section 3.3, the loss event rate of the cumulative flow is equal to the rate at which loss indications occur. To remind the reader, a lost packet is a loss indication if it is the first packet lost in a loss event (more details can be found in Section 2.4, page 21). Therefore, all subsequent lost packets within the same RTT are considered to be a part of the same loss event. In MulTFRC, the loss event rate is obtained like in TFRC, counting just one loss event per RTT. In the MulTFRC protocol, lost packets that are not loss indications are used together with loss indicators for the calculation of the number of packets lost in a loss event, which will be explained shortly. The number of packets in a loss interval is equal to the number of packets sent between two loss events. Let assume that loss interval $loss1$ starts with the packet carrying sequence number S_loss1, and packet S_loss2 is the loss indicator that starts the next loss interval. The number of packets in loss interval $loss1$ is equal to $S\_loss2 - S\_loss1$.

To obtain a smooth rate, the TFRC protocol uses the weighted average of the last k loss intervals. In [50] and [51] the value for k is set to 8; we use the same setting for MulTFRC. Let I_i ($i = 1..k$) be the number of packets in the k most recent loss intervals. The loss rate is calculated using the following formula:

$$I\_wa = \frac{\sum_{i=1}^{k} \ I\_i \ \ w\_i}{\sum_{i=1}^{k} \ w\_i} \ . \tag{4.1}$$

The weights (w_i) used for the MulTFRC implementation are: 1, 1, 1, 1, 0.8, 0.6, 0.4 and 0.2 starting from the most recent interval (this is the same as in [50]). Since the last loss interval (I_0) is not complete (more details can be found in Section 2.4, page 21), it is included in the loss interval calculation only if in this way the loss rate decreases. Hence the loss event rate is $p\_e = 1/I\_wa$.

In Section 3.3 the number of packets lost in a loss event is calculated using the approximation $j = p_r/p_e$ (the real loss rate divided by the loss event rate). Since in the MulTFRC protocol it is possible to precisely count the number of lost packets in a loss event, we use this as an input for j instead.

The theoretical arrival time of lost packets needs to be deducted by interpolation as described in [51]. As an example, we consider the packet with sequence number S_loss to be lost. S_before is the last packet with a sequence number before S_loss which arrived successfully and S_after is the first packet with a sequence number after S_loss that was received. T_before and T_after are the arrival time of packets S_before and S_after. The interpolated arrival time of packet S_loss is:

$$T\_loss = T\_before + (T\_after - T\_before)\frac{S\_loss - S\_before}{S\_after - S\_before} \ .$$

To measure j, let $t\_current$ be the interpolated time when the first lost packet in the current loss interval would arrive. We consider RTT to be the current round-trip time and $t\_newloss$ to

be the inferred time of arrival of a new lost packet. If $j\_current$ is the number of lost packets in the current loss event, the following algorithm for counting lost packets per loss event is used:

if $(t\_current + RTT) > t\_newloss$ **then**
  //packet belongs to the old loss interval
  $j\_current = j\_current + 1$
else
  //packet starts a new loss interval
  $j\_current = 1$
end if

There are different ways to estimate the number of lost packets in a loss event. It can be argued that the most recent measure is the most relevant one (since the burst size of the last loss interval influences the behavior of the cumulative flow the most), or that the average over a couple of the most recent intervals should be taken. As simulation studies have shown, taking into account the k last intervals provides a smoother behavior of MulTFRC. Hence, the same method of j samples discounting is used as for measuring the loss event rate (4.2):

$$j = \frac{\sum_{i=1}^{k} \ j\_i \ \ w\_i}{\sum_{i=1}^{k} \ w\_i} \tag{4.2}$$

This extra parameter (j) does not change the responsiveness of the MulTFRC protocol. Figure 4.1 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 Mbit/s, 20 ms RED bottleneck link with periodic loss from an ns-2 simulation. At the beginning, we set loss to 1%, at the 4th second we increased it to 10% and at the 13th second we changed it to 0.5%. To compare our protocol with TFRC, we ran five separate simulations: one TFRC flow, two TFRC flows at the same time, four TFRC flows at the same time, one MulTFRC flow with $n = 2$, and one MulTFRC flow with $n = 4$. As can be seen, the responsiveness of MulTFRC does not degrade as n increases.

We also tested the smoothness of our protocol with ns-2. We compared MulTFRC with $n = 4$ against 4 TCP flows and 4 TFRC flows; each protocol was run in a separate simulation across a 15 Mbit/s, 20 ms RED bottleneck link. We wanted to test the smoothness of protocols in a stable environment. Hence, to avoid any irregularity that can be introduced by a more complex loss model (such as random loss), we used periodic loss (the loss rate was 2%). Figure 4.2 shows that the rate of our protocol is not just significantly smoother than the cumulative rate of 4 TCPs (which is to be expected), but also smoother than the cumulative rate of 4 TFRC flows running at the same time.

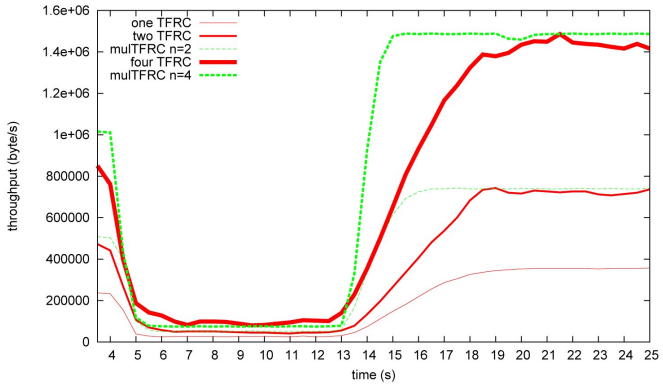


Figure 4.1: Dynamic behavior of MulTFRC: a 15 Mbit/s, 20 ms RED bottleneck link, loss rate changes over time

4.1.2 Protocol improvement

If n is a natural number, the number of "virtual" flows affected in a loss event must be between 1 and n (in algorithm 1: $j\_af = \min(\max(j,1),n);$). For $0 < n < 1$, deriving the correct number of these flows is not straightforward. We consider a flow with $0 < n < 1$ to be just like one flow that is less aggressive. Taking this assumption, the value for the number of flows affected in a loss event must be 1. In a general case, where $n \in \mathbb{R}^+$, the idea is the same. For example, if $n = 1.4$, we can consider that we have two flows: one is a normal flow ($n = 1$), and the other one is just a less aggressive flow ($n = 0.4$). Therefore, the upper limit for the number of flows that can be affected in a loss event can be obtained by rounding up ($\lceil n \rceil$) in the first line of algorithm 2. We change this line in algorithm 2 to:

$$j\_af = max(min(j, ceil(N)), 1);$$

where $ceil(n)$ gives the smallest integer greater than or equal to n.

Our equation assumes that each lost packet belongs to a different flow, which is not always true in reality. As Figure 4.23 shows, a MulTFRC flow with a large n is less affected by this assumption (additionally, MulTFRC gets slightly less throughput than TCP with $n > 70.0$ in this figure because of the exceedingly high loss rate in this extreme case (more than 11.5%); this will be discussed in more details later).

As presented at the end of subsection 3.3.1, the possibility that more than one packet belongs to the same flow can be taken into consideration for the j calculation (equation 3.52). The improved j approximation is incorporated into MulTFRC. Our studies showed that this improvement is not necessary for a larger value of n (larger than 12) and should not be used; this is due to the fact that the influence of the error introduced by this assumption is proportionally

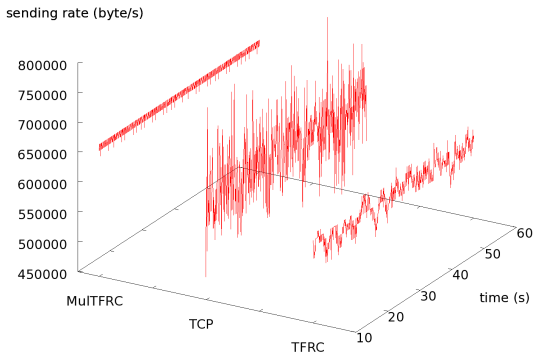


Figure 4.2: Rate smoothness of MulTFRC, TCP and TFRC: a 15 Mbit/s, 20 ms RED bottleneck link with periodic loss

smaller with an increase of the number of flows. Figure 4.23 also shows TCP flows compared with MulTFRC flows when the improved j approximation is used. Here the difference between the throughput of TCPs and MulTFRC is smaller.

Algorithm 2 should be altered by adding the following part at the beginning of the algorithm and changing the first line:

if $(n < 12)$ **then**
  $j\_af = n * (1 - (1 - 1/n)^j)$;
else
  $j\_af = j$;
end if
$j\_af = max(min(j\_af, ceil(n)), 1)$;

The complete algorithm (algorithm 3) with all changes is given at the end of this thesis, in Appendix A.

4.2 Evaluation with simulations

The behavior of MulTFRC was verified using simulations. A broad range of network conditions was used. The aggression parameter was varied taking values up to 100 and thereby not just values that are natural numbers, but also real numbers as well as values between zero and one.

Furthermore, special attention was given to comparing MulTFRC with similar protocols, like MulTCP [36], CP [101] and Stochastic TCP [59].

We start with the validation of MulTFRC in isolation. We will observe MulTFRC under the influence of different loss models as well as its behavior in a network topology without artificial loss.

4.2.1 MulTFRC in isolation

In this subsection, we will validate the behavior of MulTFRC in isolation. Similar to the validation of the equations (Sections 3.2.4 and 3.3.4), different network conditions were considered. The same simulation setups as in the corresponding sections of the model validation were used. To remind the reader, the dumbbell topology was used, only in this case, we had a single MulTFRC flow running between a sender (denoted by "S" in Figure 3.5) and a receiver (denoted by "R" in Figure 3.5). The link X-Y from Figure 3.5 was shared by multiple TCP connections and therefore it was called the "shared link". Since MulTFRC does not share the path with other flows, we will also refer to this link as the "middle link". In all simulations from this subsection, no additional background traffic was used. The characteristics of the middle link and MulTFRC's parameter n were varied. We simulated random loss on the middle link (each packet is lost with the same probability), burst loss on the middle link as well as a network topology without artificial loss. In all simulations, the throughput on the S-X (Figure 3.5) link was measured — the rate at which packets leave the sender, thereby the fact that TCP retransmits packets does not have an influence on the results. This is the same as in Subsections 3.2.4 and 3.3.4. The first 50 seconds were omitted for throughput measurements because we wanted to avoid that the startup phase influences them. MulTFRC used a packet size of 1032 bytes (this included 1000 bytes of data and a 32-bytes header). The same data size of 1000 bytes was used in corresponding simulations with TCP flows (Subsection 3.3.4). We chose this value for consistency and easier comparison of MulTFRC and TFRC because the same value was used in [50]. The throughput of a MulTFRC flow was compared with the throughput of n TCP flows obtained in the corresponding separate simulations as well as with the model used for the design of MulTFRC (the model with the cumulative flow loss measure, Section 3.3). The results for the TCP throughput and the model are the same as in the corresponding figures from Section 3.3.4. For more details about the way they were obtained, the reader is referred to that subsection.

Simulations with random loss

First we present a simulation setup with a random artificial loss introduced on the middle link. The same topology was used in the part of Subsection 3.3.4, marked as "Simulations with random loss". To remind the reader, we used a random loss model which causes the probability that a packet is lost to be the same for all packets. The access links had a bandwidth of 10 Gbit/s and a delay of 1 ms, whereas the bandwidth and delay of the middle link were 1 Gbit/s and 30 ms, respectively. We also simulated a delay of 100ms on the middle link. Both the RED and DropTail queuing mechanisms were used on the middle link. Each simulation

lasted 460 seconds. The first 50 seconds were omitted for the throughput calculation because the startup phase produces a different behavior which is not of interest here.

MulTFRC's parameter n was varied, and it had values 5, 10, 50 and 100. The throughput of a MulTFRC flow with parameter n was measured and compared with the throughput of n TCP flows. We also compared the MulTFRC throughput with results of the equation with the cumulative flow loss measure. The last two are the same as in figures 3.19, 3.20, 3.21, 3.22, 3.23 and 3.24. Since the equation without the improved j calculation was used for these figures, we start this discussion by looking at MulTFRC without this improvement and in the end we examine the protocol including the improved j calculation.

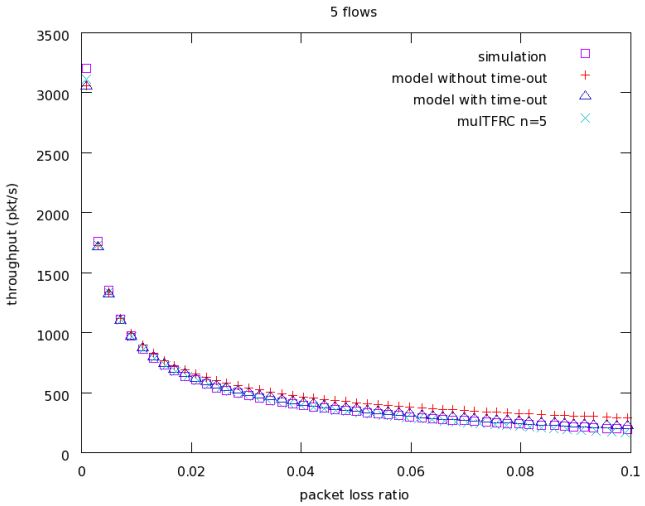
First we will observe simulations with a RED queue on the middle link. The RED parameters were configured as in subsection 3.3.4 ($q\_weight = -1$, $thresh = 0$, and $maxthresh = 0$ were set for automatic configuration and the "gentle" mode was used). Figures 4.3 and 4.4 show results for 30 ms middle link delay and the results with the middle link having a delay of 100 ms are shown in Figure 4.5. The throughput obtained by MulTFRC corresponds to the one of parallel TCP flows. The same results are seen irrespective of parameter n. MulTFRC always has a slightly lower rate than TCP flows running under the same condition.

For loss rates smaller than 2.7%, the difference between MulTFRC and TCPs is always less than 1.1% of the TCP flows' throughput. Only for the very low loss ratio of 0.1% this difference is slightly higher; MulTFRC has 2.7%, 3.7%, 4.8% and 4.9% lower rates than TCP for n equal to 5, 10, 50 and 100, respectively. This is due to a smaller rate calculated by the model. For this loss ratio, the rate of MulTFRC with n equal to 5 is between the rate obtained by parallel TCP flows and the equation; it is 2% higher than the one estimated by the model and 2.7% lower than the throughput of the TCP flows. Also for the other values of parameter n and a loss rate of 0.1%, MulTFRC obtains a throughput which is between the TCP flows' throughput and the model but it is closer to the value predicted by the model (the throughput of MulTFRC is 1% greater than the throughput estimated by the model).

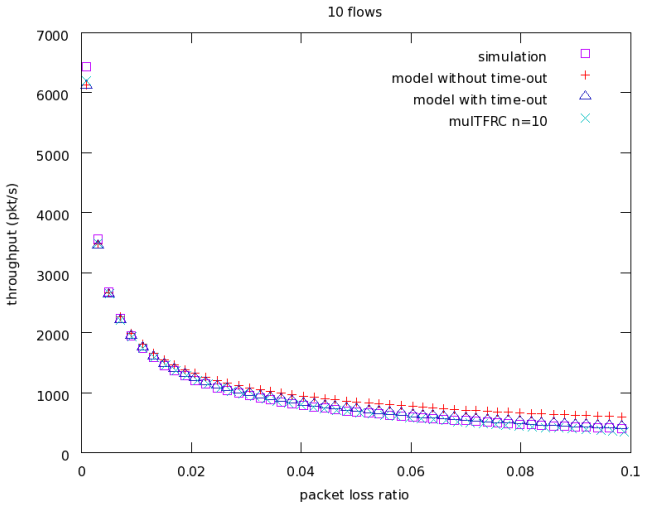
As the loss rate is increased, the difference between MulTFRC and TCP slightly grows with MulTFRC having lower rates. The difference only exceeds 5% of the TCP flows' throughput for a loss rate larger than 5% (this is the same for all values of parameter n), and the MulTFRC flow has a more than 10% lower rate than the TCP flows only for a high loss rate of more than 6.9%.

Changing the middle link delay to 100 ms gave even better results, as shown in Figure 4.5. The difference between MulTFRC and TCPs never exceeds 4.5% of the TCP flows' throughput, even for a high loss rate of 10%. For parameter n equal to 10 and 50 and a loss rate of 0.1%, MulTFRC has a 3.6% and 4% lower throughput than TCP. Further this difference decreases, and it is less than 1% (1% of the TCP flows' throughput) for a loss ratio smaller than 1%. From loss ratios between 1% and 10%, the difference varies between 0.1% and 4.5% of the TCP flows' throughput.

As discussed in the previous section, the improved j calculation should be used for $n \leq 12$. Figure 4.6 shows results with this change. The improved j calculation was introduced so that

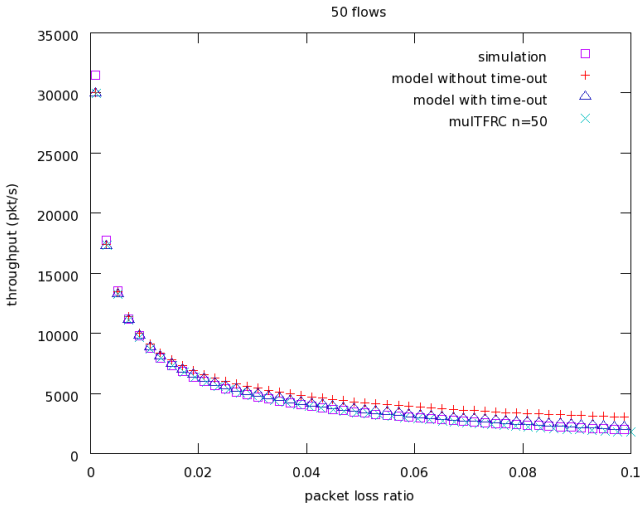


(a)

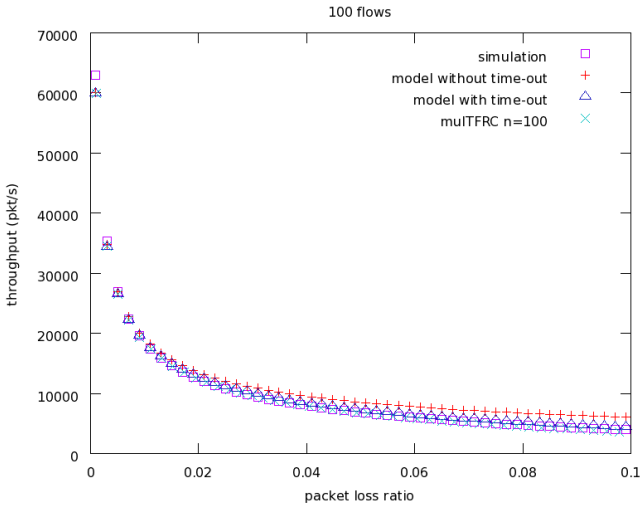


(b)

Figure 4.3: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the version without the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), RED queue, 30ms bottleneck delay

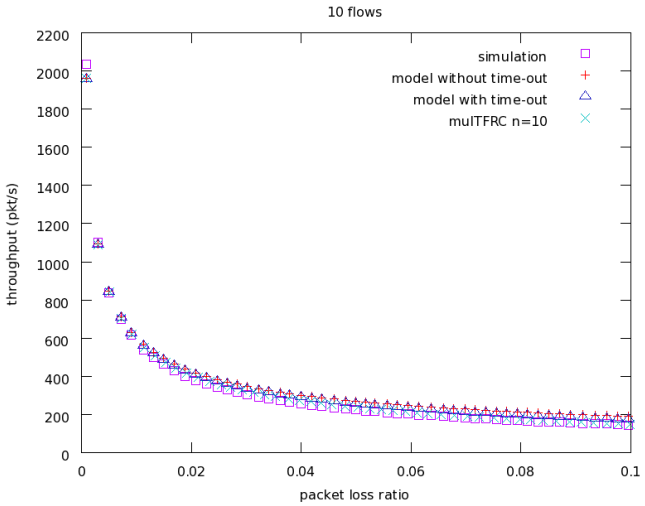


(a)

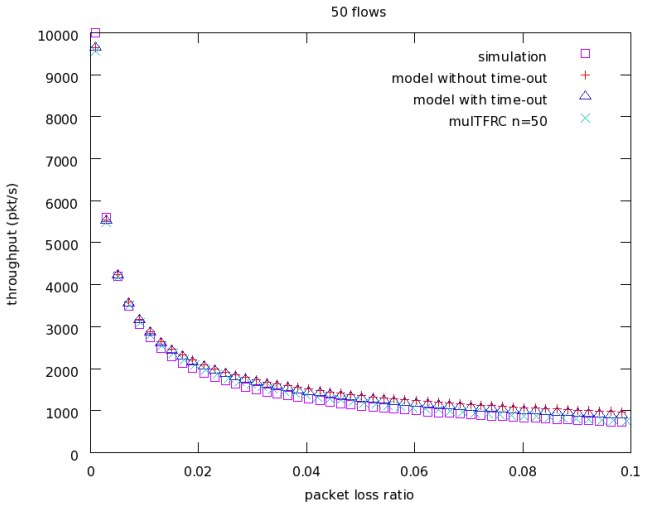


(b)

Figure 4.4: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 50 and 100) with the throughput of 50 and 100 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), RED queue, 30ms bottleneck delay

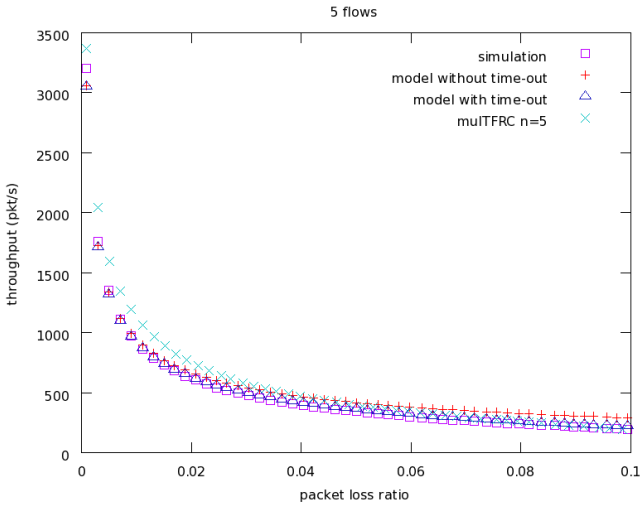


(a)

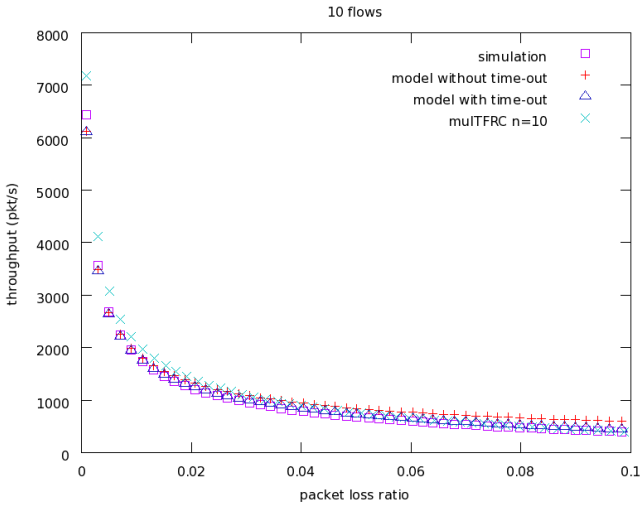


(b)

Figure 4.5: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 10 and 50; without the improved j calculation) with the throughput of 10 and 50 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), RED queue, 100ms bottleneck delay

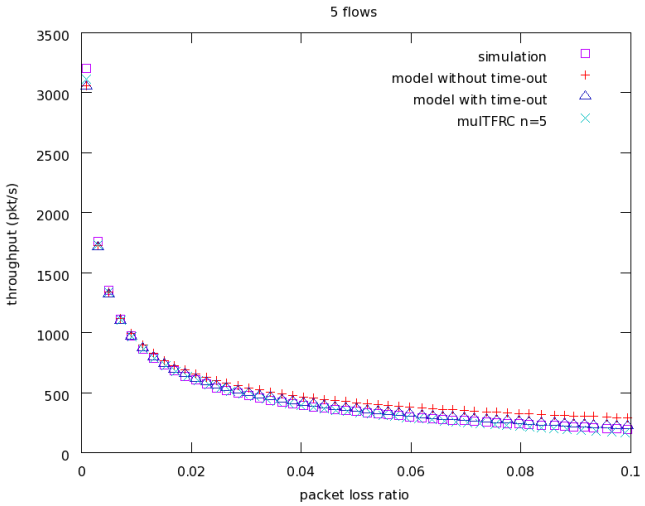


(a)

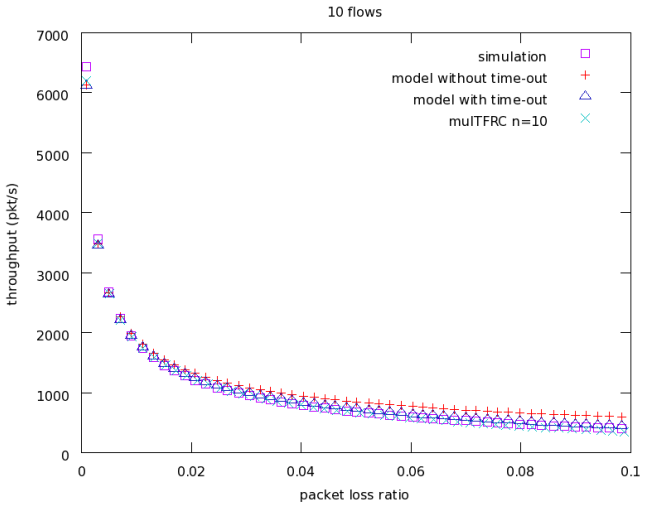


(b)

Figure 4.6: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the version with the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), RED queue, 30ms bottleneck delay

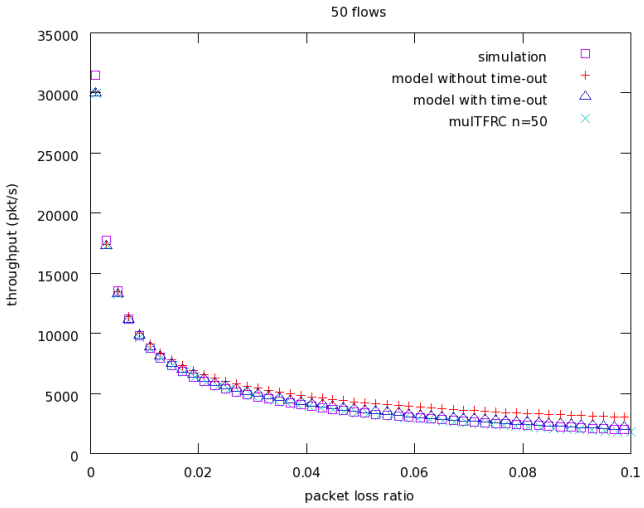


(a)

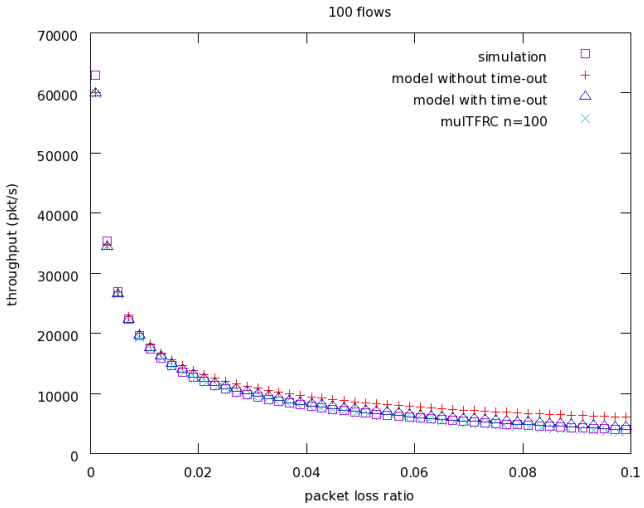


(b)

Figure 4.7: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the version without the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), DropTail queue, 30ms bottleneck delay

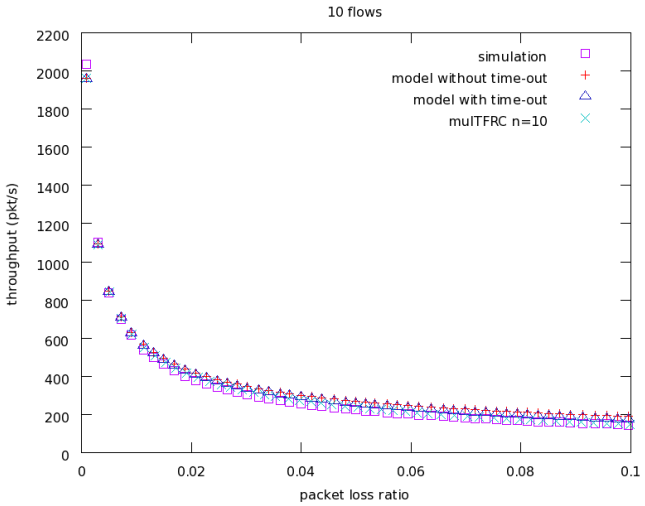


(a)

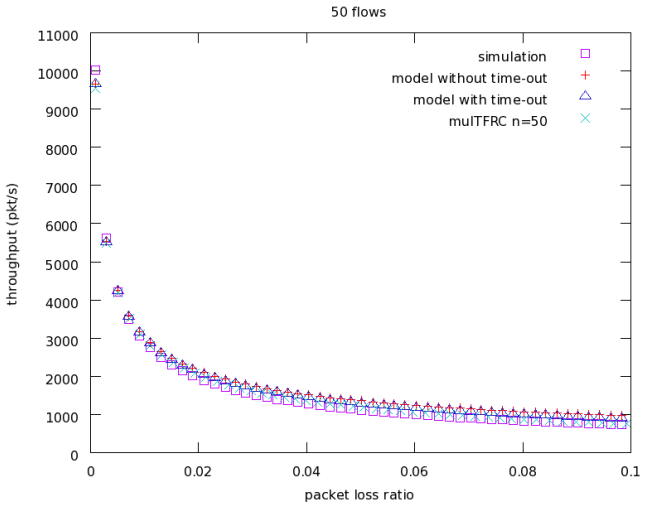


(b)

Figure 4.8: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 50 and 100) with the throughput of 50 and 100 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), DropTail queue, 30ms bottleneck delay



(a)



(b)

Figure 4.9: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 10 and 50; the version without the improved j calculation was used) with the throughput of 10 and 50 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations with random loss on the middle link (each packet is lost with the same probability), DropTail queue, 100ms bottleneck delay

MulTFRC can better cope with a more correlated loss, assuming that more than one lost packet can belong to the same virtual flow. On the other hand, the simulations presented here introduce a non-correlated 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 achieves a higher rate than TCP flows. As the loss rate grows, the probability that more than one lost packet belongs to the same flow grows and the difference between TCP flows and MulTFRC becomes smaller. As seen in Figure 4.6, in case of a low loss rate, MulTFRC with the improvement obtains an even 15% higher rate than the corresponding TCP flows. This difference decreases as the loss rate grows. For loss rates between 5% and 8%, the difference is in the range between 0.1% and 5% of the TCP throughput. For a higher loss rate, MulTFRC with the improved j calculation gives almost the same results as without this improvement. The same observations were made for the middle link delay of 100 ms and MulTFRC with the improved j calculation.

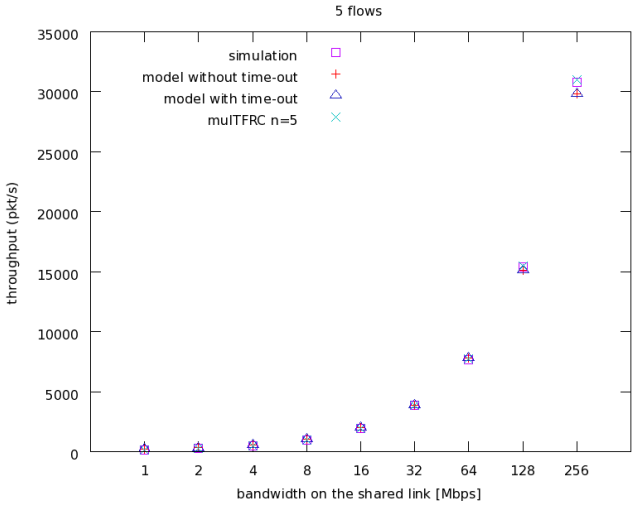
As already discussed in Subsection 3.3.4, simulations with a DropTail queue on the middle link show the same results; this is because the loss is only produced by the loss model and the used queuing mechanism does not influence the results. For completeness, the results with the DropTail queuing mechanism are presented in Figures 4.7, 4.8 and 4.9.

Simulations with loss from a DropTail (FIFO) queue and a RED queue

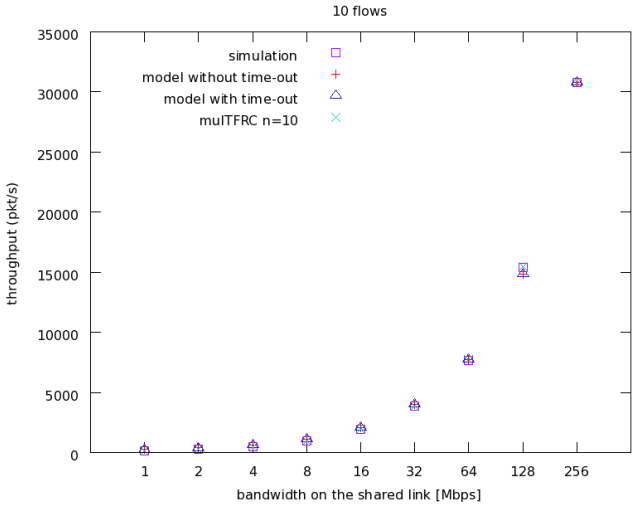
For these simulations, we used the same simulation setup as for the models validation without artificial loss; we are referring to the validation part marked by "Simulations with loss from a DropTail (FIFO) queue and a RED queue", subsection 3.2.4, page 80. For convenience, we will repeat in short the network topology used for these simulations: the access links had a delay of 1 ms. The capacity of the access links was 1 Gbps. The middle delay was 30 ms and the bottleneck capacity was changed to cause a varying amount of loss. It had the values 1 Mbps, 2 Mbps, 4 Mbps, 8 Mbps, 16 Mbps, 32 Mbps, 64 Mbps, 128 Mbps and 256 Mbps. Simulation setups with a DropTail queue and a RED queue on the middle link were used. The duration of each simulation was 600 seconds.

First, we will observe simulations with a DropTail queue on the middle link. The queue length of the DropTail queue was equal to the bandwidth $\times$ delay product. Figures 4.10 and 4.12 show results for parameter n equal to 5, 10, 50 and 100. We compared MulTFRC with the throughput of parallel TCP flows and the equation. The results of the simulations with TCP flows and the equation outcome is the same as in Figures 3.25 and 3.26. Here we show the results depending on the middle link capacity because the TCP flows and MulTFRC experience a slightly different packet loss ratio. MulTFRC had a high loss ratio because it normally sends at a rate close to the bottleneck link bandwidth and it probes the network more often than parallel TCP flows. Both MulTFRC and TCP obtained the same rate and high link utilizations were seen (around 99%). As shown in Figure 4.11, MulTFRC with the improved j calculation gives the same results.

The second set of simulation had a RED queue implemented on the middle link. The RED parameters recommended in [3] were used: we let the algorithm implemented in ns-2 automati-

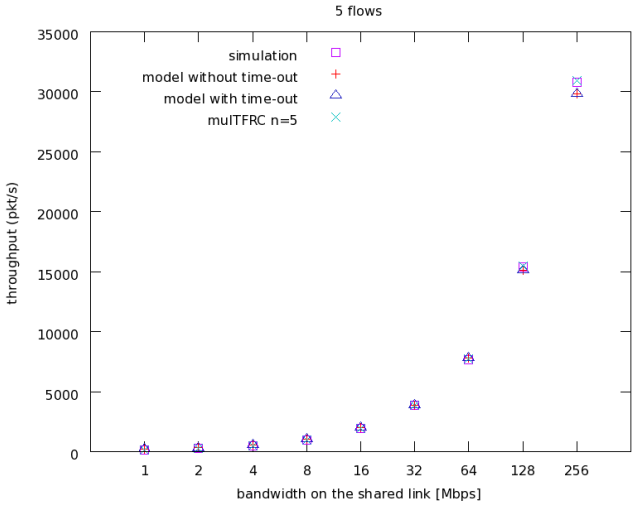


(a)

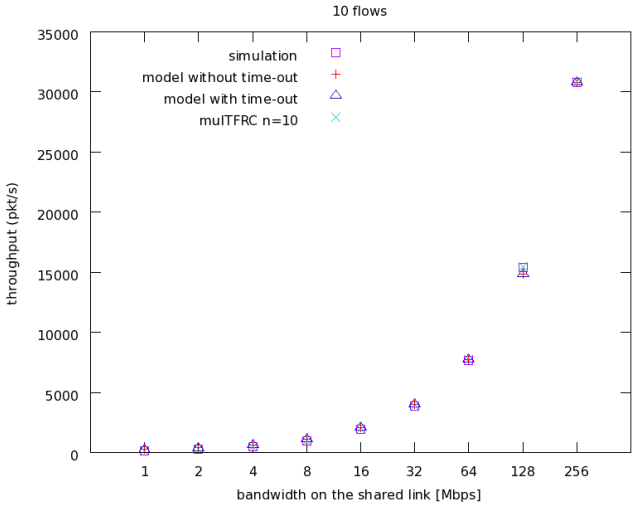


(b)

Figure 4.10: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the version without the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations without artificial loss on the middle link, DropTail queue, 30ms bottleneck delay

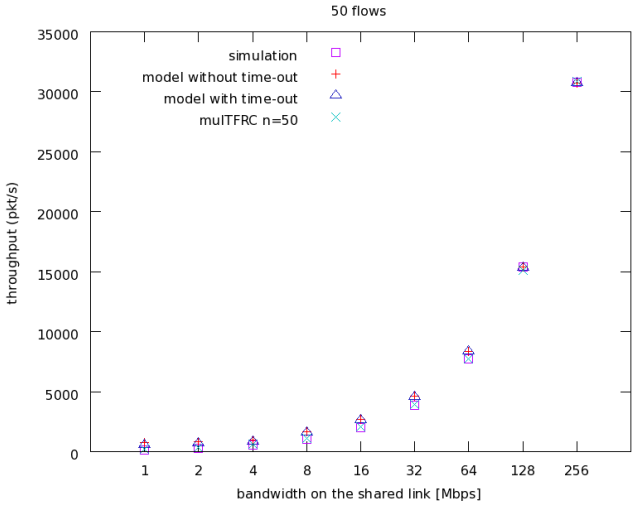


(a)

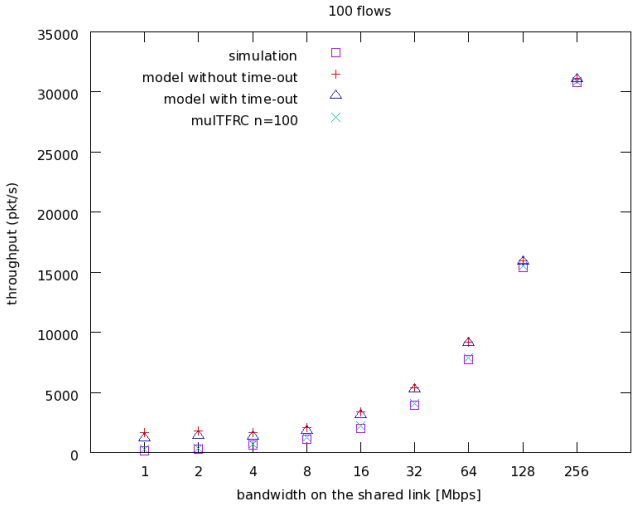


(b)

Figure 4.11: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations without artificial loss on the middle link, DropTail queue, 30ms bottleneck delay

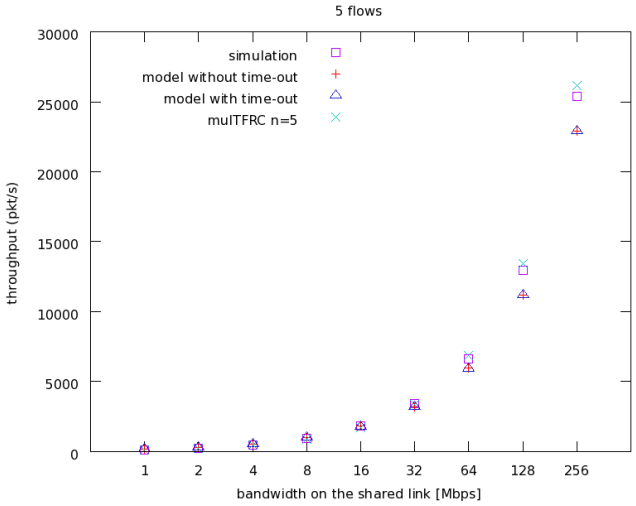


(a)

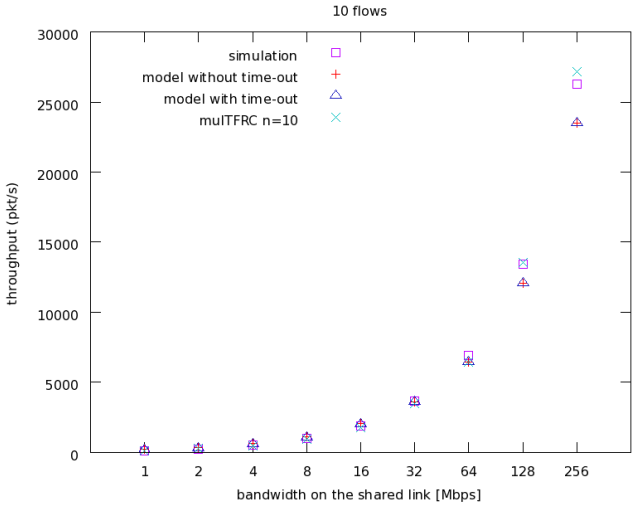


(b)

Figure 4.12: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 50 and 100) with the throughput of 50 and 100 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations without artificial loss on the middle link, DropTail queue, 30ms bottleneck delay

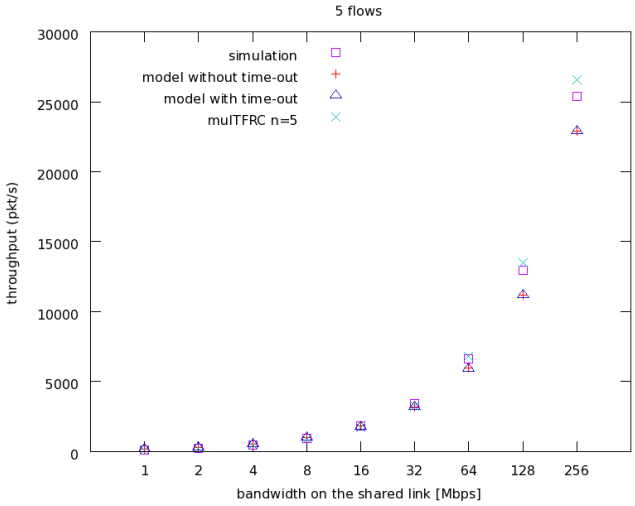


(a)

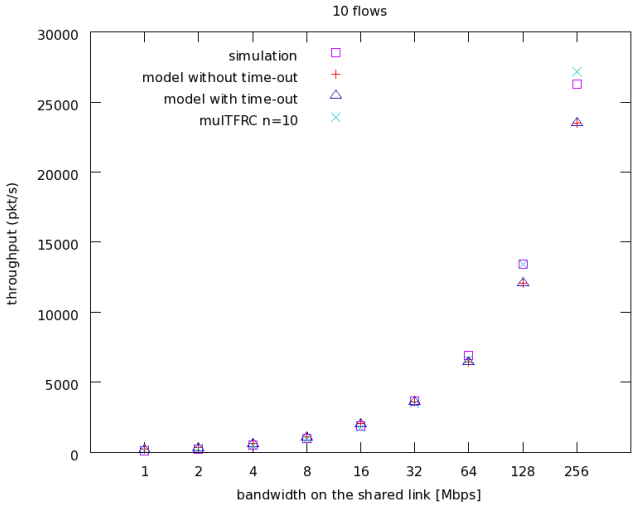


(b)

Figure 4.13: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the version without the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations without artificial loss on the middle link, RED queue, 30ms bottleneck delay

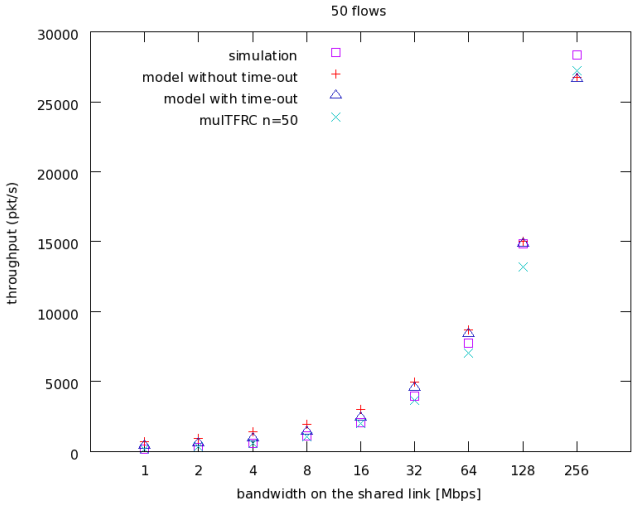


(a)

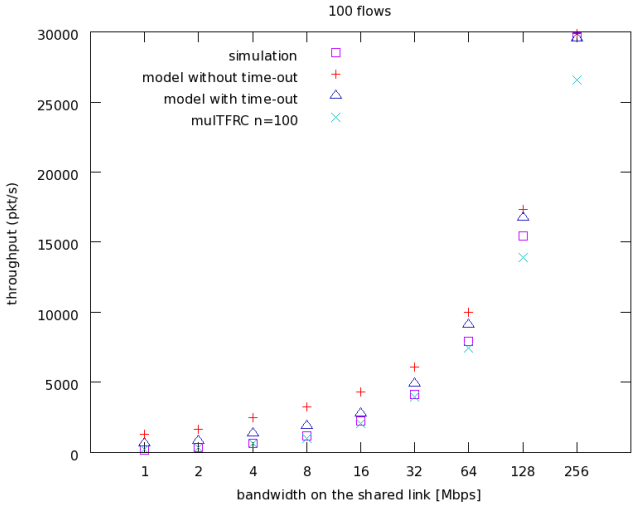


(b)

Figure 4.14: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 5 and 10; the improved j calculation was used) with the throughput of 5 and 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations without artificial loss on the middle link, RED queue, 30ms bottleneck delay



(a)



(b)

Figure 4.15: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 50 and 100) with the throughput of 50 and 100 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: simulations without artificial loss on the middle link, RED queue, 30ms bottleneck delay

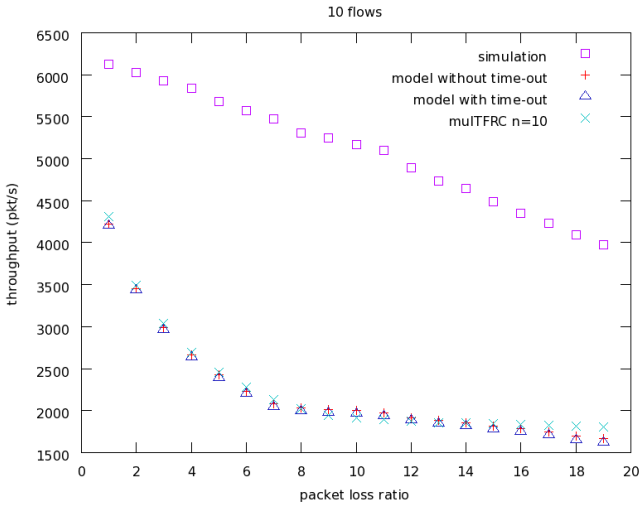
cally configure them — $q\_weight = -1$, $thresh = 0$, and $maxthresh = 0$ were set for automatic configuration and the "gentle" mode was used. Other network parameters were the same as in the previous simulations with a DropTail queue.

The RED queue has a different influence on the MulTFRC flow than on the TCP flows. A MulTFRC sender sends packets spread over a round-trip time and TCP sends packets in bursts. For a small value of parameter n, MulTFRC achieves a higher throughput than TCP flows and a better link utilization, as shown in Figure 4.13. The throughput of MulTFRC is about 2% to 4% higher than the throughput of the TCP flows for a bottleneck capacity of 128 Mbps and 256 Mbps. MulTFRC with the improved j calculation shows the same results (Figure 4.14). For other values of the bottleneck capacity, MulTFRC has a between 0.7% and 7% lower rate than TCP for all values of parameter n. On the other hand, a large n and a bottleneck capacity of 128 Mbps and 256 Mbps causes MulTFRC to gain a lower rate than TCP flows (Figure 4.15). This is due to a small queue which tries to achieve a target delay of 0.005s, and MulTFRC, which is sending with a rate close to the bottleneck bandwidth, experiences loss in a number of consecutive round-trip times 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 [50]. The results are quite different: in this case, MulTFRC obtains a higher rate even for large values of n, and it is able to saturate the middle link.

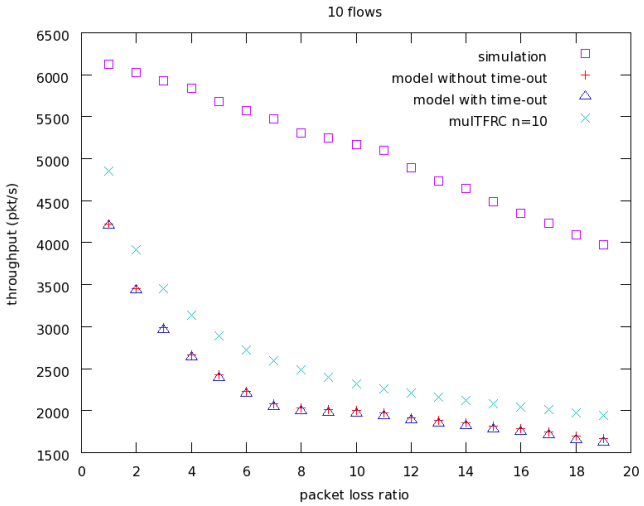
Simulations with burst loss

The following part shows results of simulations where a burst loss model was introduced on the middle link. They correspond to the simulations from the part "Simulations with burst loss" in Subsection 3.3.4. These simulations had the same network setup as the simulations with random loss, only here a burst loss model was used on the middle link. This loss model always drops a certain number of packets in sequence, and this number is the parameter we varied. The loss burst occurred with a probability of 0.1% and each packet had the same chance to be the start of a burst. Each simulation had a duration of 600 seconds. Since the simulation results did not depend on the used queuing mechanism, we will only show the results with a RED queue on the middle link.

Similar to the equation in such a simulation setup, MulTFRC does not match the throughput of parallel TCP flows, as shown in Figure 4.16. Some shortcomings of the ns-2 simulator are to be seen here. There is no multiplexing between flows in such a scenario and therefore a burst loss occurrence mostly affects a single flow (for more details the reader is referred to Subsection 3.3.4). Such an extreme behavior is not to be expected in the Internet. Even burst losses are rather rare in the Internet (as described in [25], 83% of losses are just drops of a single packet). Our equation assumes non-bursty loss. We modeled flows that are mixed in a round robin fashion and such a composition of flows would be affected differently by bursty loss, i.e. the loss burst would affect multiple flows. The equation does not estimate the throughput of TCP flows well under such circumstances.



(a)



(b)

Figure 4.16: Comparing the throughput of a single MulTFRC flow (it had parameter n set to 10) with the throughput of 10 TCP flows and the throughput estimated using the model with the cumulative flow loss measure: (a) shows MulTFRC without the improved j calculation and (b) shows MulTFRC with the improved j calculation; simulations with burst loss on the middle link, RED queue, 30ms bottleneck delay

As the MulTFRC behavior mainly depends on the equation, the shortcomings of the model are inherited. The throughput of MulTFRC under a loss burst of different lengths is shown in Figures 4.16 (a). MulTFRC obtains the same throughput as estimated by the equation but its throughput is smaller than the throughput of parallel TCP flows running under the same conditions. Since an occurrence of bursty loss affects only a single TCP flow, even the improved j calculation does not give satisfying results (Figure 4.16 (b)). This improvement assumes that a lost packet belongs to each flow with the same probability, but the ns-2 simulator produces burst loss which almost always drops packets belonging to a single flow.

4.2.2 MulTFRC responsiveness

In section 4.1 we showed the responsiveness of MulTFRC when network conditions change. There, a periodic loss model was used and the loss probability of the model was changed during a simulation. Here we further investigate whether cross-traffic produces the same effect. We study the behavior of MulTFRC in the presence of non-reactive traffic as well as reactive traffic. MulTFRC with the improved j calculation was used in these simulations.

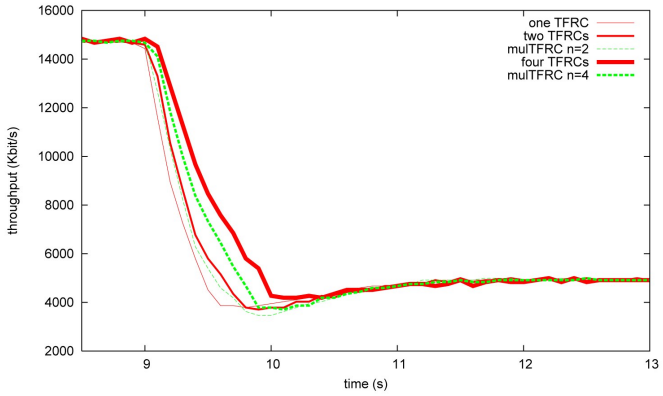


Figure 4.17: MulTFRC responsiveness: UDP traffic started at the 9th second; UDP had a constant bit rate of 10Mbps; the shared link bandwidth was 15Mbps and the delay was 20ms; the shared link implemented the DropTail queuing mechanism

First, we consider non-reactive traffic — UDP traffic. The behavior of MulTFRC was compared with the behavior of a number of TFRC flows. Separate simulations were run for MulTFRC sharing the link with a UDP flow (two separate scenarios are investigated — MulTFRC with $n = 2$ and MulTFRC with $n = 4$) and for TFRC flows competing with UDP (we ran separate simulations for one, two and four TFRC flows). The common link had the bandwidth and delay equal to 15 Mbps and 20 ms, respectively. All links implemented the DropTail queue mechanism. There was no artificial loss on the shared link. Three scenarios were simulated.

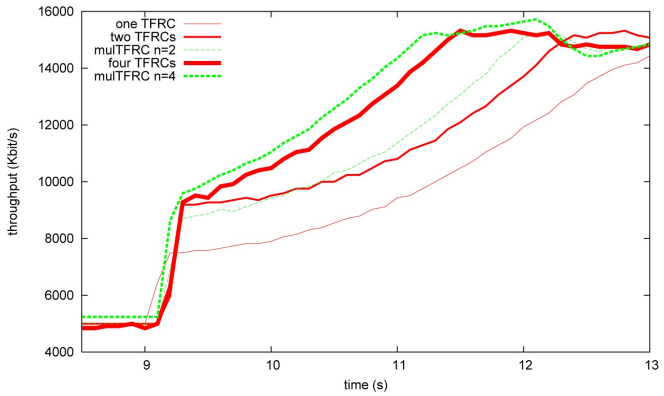


Figure 4.18: MulTFRC responsiveness: UDP traffic stopped at the 9th second; UDP had a constant bit rate of 10Mbps; the shared link bandwidth was 15Mbps and the delay was 20ms; the shared link implemented the DropTail queuing mechanism

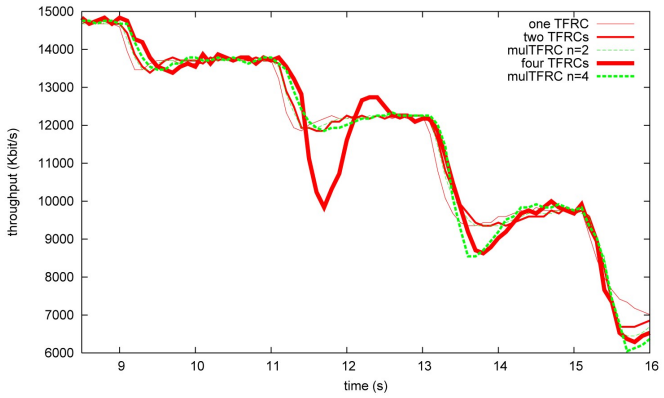


Figure 4.19: MulTFRC responsiveness: UDP traffic started at the 9th second with a constant bit rate of 1Mbp, further UDP increased its rate by 2.5Mbps every 2 second; the shared link bandwidth was 15Mbps and the delay was 20ms; the shared link implemented the DropTail queuing mechanism

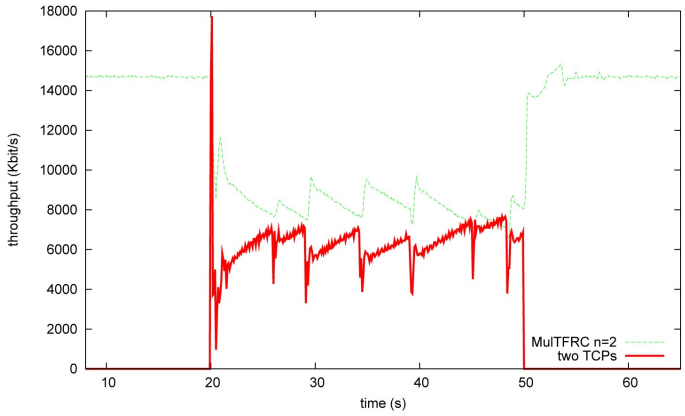


Figure 4.20: MulTFRC responsiveness: MulTFRC with parameter n equal to two and two TCP flows; because of a route change, converged TCP enters the shared link at the 10th second and leaves the shared link at the 50th; the shared link implemented the DropTail queuing mechanism

In Figure 4.17, the results of the first scenario are shown. Here a UDP flow sending at 10 Mbps started after MulTFRC or TFRC had been active for some time and had converged; it started at the 9th second. MulTFRC with parameter n reacts faster than n TFRC flows. For $n = 2$, the difference is rather small but, for $n = 4$, MulTFRC decreases its rate to the value of the new appropriate rate after 0.8 seconds and 4 TFRC flows need one second.

The second scenario shows the inverse case — MulTFRC (TFRC) and the UDP flow sending at 10 Mbps started at the same time, and at the 9th second the UDP flow terminated. The sending rate of MulTFRC and TFRC flows over time are depicted in Figure 4.18. After the first fast increase MulTFRC and TFRC show a slower rate increase which is due to a slow increase of the last loss interval duration. MulTFRC increases its rate faster than the corresponding number of TFRC flows.

For the last scenario, the UDP traffic started at the 9th second with a rate of 2.5Mbps and its rate was increasing every 2 seconds by 2.5Mbps., as shown in Figure 4.19. MulTFRC reacts as fast as the corresponding number of TFRC flows. After the reduction, MulTFRC immediately obtains a stable rate and the rate of the TFRC flows fluctuates. After the UDP rate is increased at the 11th second, 4 TFRC flows reduces their sending rate drastically. This is probably because the first loss caused by the newly introduced traffic happened at the beginning of a loss interval; therefore, this loss interval was very short and it caused the average loss rate to be high. A similar effect is to be seen after the 13th second. In this case, both MulTFRC with $n = 4$ and the 4 TFRCs experience this effect.

Furthermore, we investigate how MulTFRC responds to reactive traffic, like TCP. Instead of introducing TCP flows which start slowly with sending a packet per round-trip time and an

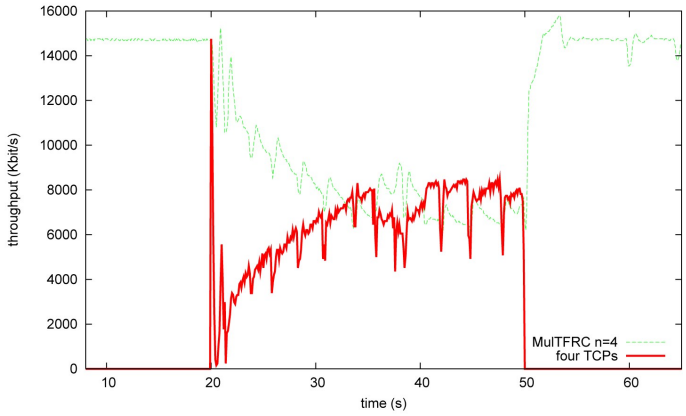


Figure 4.21: MulTFRC responsiveness: MulTFRC with parameter n equal to four and four TCP flows; because of a route change, converged TCP enters the shared link at the 10th second and leaves the shared link at the 50th; the shared link implemented the DropTail queuing mechanism

exponential rate increase, it is more interesting to see what happens when MulTFRC and TCP are already converged and MulTFRC and TCP have the same rate before an interaction. To achieve this, we simulated a network with a route change. First, TCP flows and a MulTFRC flow ran through two separate links with the same characteristics. Each link had a bandwidth of 15 Mbps and 20 ms delay. A DropTail queue was used on all links. The length of the queue was the same as in [50] (the length was three times the bandwidth $\times$ delay product). After the TCP flows and MulTFRC had converged, a route of the TCP flows was changed, at the 20th second. Instantly, TCP flows and MulTFRC flow shared the same link and both traffic reduced its rate. At the 50th second the route was changed again, TCP flows left the shared link and MulTFRC increased its rate again. The behavior of MulTFRC and TCP flows over time can be seen in Figures 4.20 and 4.21. In Figure 4.20, MulTFRC had parameter n set to 2 and there were two TCP flows. In the second figure, the interaction of a MulTFRC flow with $n = 4$ and four TCP flows is shown.

As one of the goals of MulTFRC is to achieve a smoother rate it reacts slower than TCP. As seen in the figures, the TCP flows reacted faster and probably some of the flows experienced a timeout and reduced their rate to one packet per RTT. Since the queue was large, the RTT was increased and the convergence time was longer. With a smaller queue, the convergence time would be much shorter.

4.2.3 MulTFRC vs. TCP

The main goal that we wanted to achieve with MulTFRC is n-TCP-friendliness. Therefore we will investigate the behavior of the protocol in the presence of TCP flows and examine its n-TCP-friendliness. First, we will explain the used simulations setups and the used normalization for this study.

Simulation setup

The evaluation was done using the ns-2 simulator [4] (version 2.33). The standard network configuration for testing transport layer protocols, the so-called "dumbbell" topology, was used. This is the same topology as the one used in Section 3.2.4, only in this case one of the connections runs the MulTFRC protocol; this is shown Figure 4.22. In each simulation n_{TCP} standard TCP flows (TCP Sack was used) were run together with a MulTFRC flow having the aggression parameter set to $n_{MulTFRC}$. Also here, packets carried 1000 bytes of data (this is the same as in [50]; we chose this value for easier comparison with the original TFRC). To study the influence of the MulTFRC protocol on the standard TCP protocol in a broad range of network conditions, we ran simulations with a wide set of bottleneck capacities and with both RED and DropTail queues. The link capacities were chosen in such a way that the bottleneck on the middle link is ensured. The bottleneck capacity and parameters n_{TCP} and $n_{MulTFRC}$ were varied. For this set of simulations no artificial loss was used.

In all simulations the first 15 seconds are omitted for the throughput measurements; the simulations were run long enough to obtain the steady-state behavior of both protocols (at least 5500 RTTs). The throughput was calculated by looking at the number of packets arriving at the receiver, irrespective of their sequence number and therefore, packet retransmissions of TCP does not influence the results.

Normalization

Since in simulations a certain number of TCP flows are competing against one MulTFRC flow, for an easier comparison of the behavior of each protocol, a normalization, defined in the following text, is applied. The MulTFRC flow is considered to be a single flow that consists of a number of "virtual" flows. Therefore, for the validation presented here, the throughput of one "virtual" flow is used; it is calculated as the throughput of MulTFRC divided by the MulTFRC aggression parameter. In all figures the normalized average throughput of the TCP flows and the normalized throughput of a single "virtual" MulTFRC flow are shown. Let n_{TCP} be the number of TCP flows traversing the bottleneck link and let $n_{MulTFRC}$ be the aggression parameter of the MulTFRC flow. Further, we denote by B_i the throughput of the i-th TCP flow, where $i = 1..n_{TCP}$, by $B_{MulTFRC}$ the throughput of the MulTFRC flow (the throughput of one "virtual" flow is $B_{MulTFRC}/n_{MulTFRC}$) and by B_{norm} the fair throughput of each single flow (respectively "virtual" flow in the case of MulTFRC) on the given bottleneck link. If the bandwidth of the bottleneck link is BW, B_{norm} is defined as:

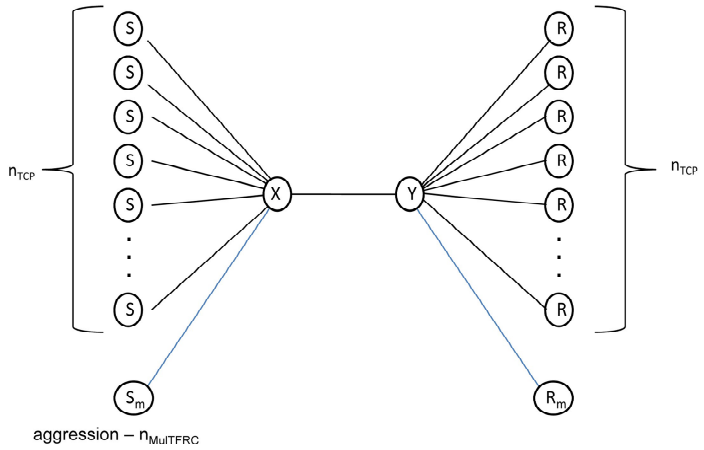


Figure 4.22: Dumbbell topology with MulTFRC

$$B_{norm} = \frac{BW}{n_{TCP} + n_{MulTFRC}} \ .$$

The normalized average throughput of the TCP flows is calculated using the following formula:

$$B_{TCP_{norm}} = \frac{\frac{\sum_{i=1}^{n_{TCP}} B_i}{n_{TCP}}}{B_{norm}} \ .$$

The normalized throughput of a single "virtual" MulTFRC flow is:

$$B_{MulTFRC_norm} = \frac{\frac{B_{MulTFRC}}{n_{MulTFRC}}}{B_{norm}} \ .$$

This normalization is applied in all figures except when stated otherwise.

The aggression parameter $n \in \mathbb{N}$

The network configurations used in this subsection are based on the simulations used in [50], i.e., the same network parameters were used. All queues, except for the queue of the bottleneck link, implement the DropTail queuing mechanism. Access links have a 10 Gbit/s capacity and 2 ms delay on the left (sender) side of the bottleneck link, and 1 ms on the right-hand side of the bottleneck link (Figure 4.22, links from node Y to the receivers). The characteristics of the bottleneck link were varied in these tests, but care was taken that this link was always the real bottleneck of the network.

At the end of the previous section a possible refinement of the j calculation has been discussed and hence the evaluation of the MulTFRC protocol starts with a comparison between the old and the new j calculation, shown in Figure 4.23. Here, MulTFRC and TCP flows shared a 32 Mbit/s, 20 ms RED bottleneck link. The RED parameters were the same as in [50]. The *gentle* mode was enabled on the bottleneck queue and the buffer capacity was equal to three times the bandwidth × delay product; the "minthresh" and "maxthresh" parameters of RED were set to one tenth and one third of this product respectively (the same as in [50]). For values of n smaller than 13, the improved j calculation shows an advantage. This figure also shows that MulTFRC behaves "n-TCP-friendly" even for a large value of n. As it can be seen, the difference between the normalized throughput of the TCP flows and the MulTFRC flow exceeds 0.1 (which means that MulTFRC has a more than 10% lower throughput than TCPs) only for n greater than 70. For this large number of flows (70 TCP flows and a MulTFRC flow behaving as 70 TCP flows) a very large loss rate, greater than 11.5%, was seen.

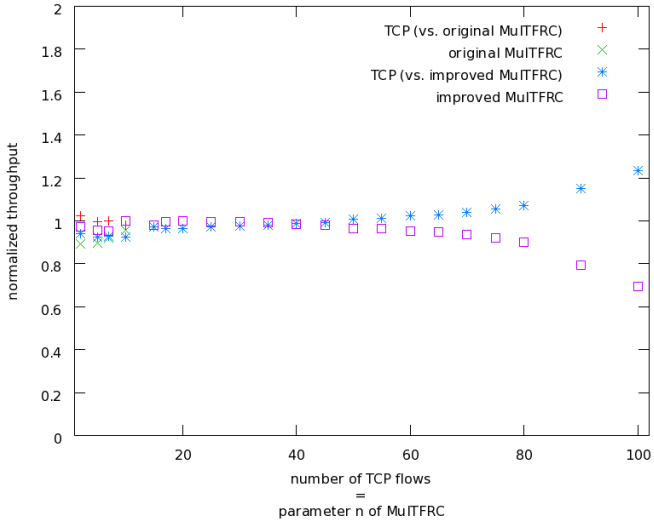
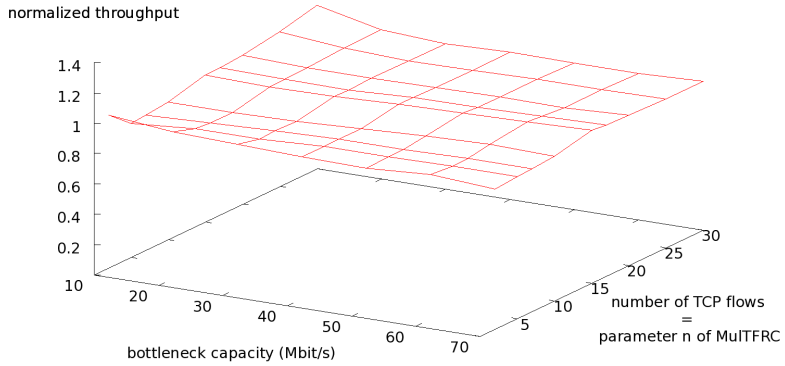
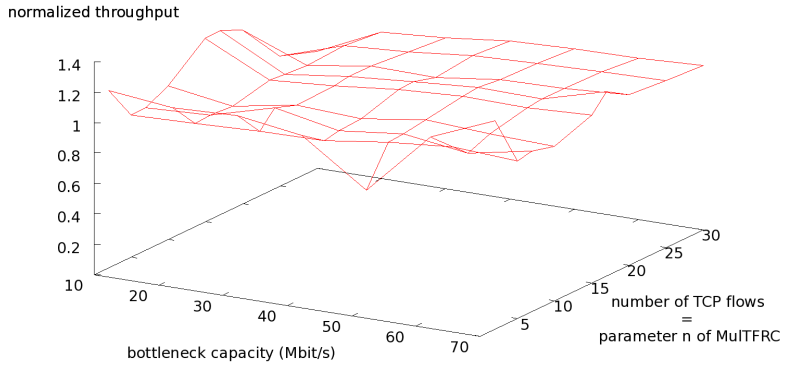


Figure 4.23: TCP-friendliness of MulTFRC with old and new calculation of j (which only differs for $n < 13$): the bottleneck link capacity was 32 Mbps and the link delay was 20 ms; the difference between TCPs and MulTFRC exceeds 0.1 only for $n \geq 70$

Furthermore, we evaluated the MulTFRC protocol with a range of values for the aggression parameter — n. In each simulation a MulTFRC flow shared the bottleneck link with several TCP flows; the number of TCP flows was the same as MulTFRC's parameter — n. Only MulTFRC with the improved j calculation was used. For the first set of simulations, the bottleneck link implemented the RED queue mechanism. The bottleneck link delay was 20 ms and the capacity was changed from 10 Mbit/s to 70 Mbit/s. The RED queue parameters were the same as in the previous simulation. The impact of MulTFRC on TCP with a RED queue is shown in Figure 4.24 (a). The capacity is shown on the x-axis and parameter n is on the y-axis. The figure depicts the normalized throughput of TCP; ideally, these values should be 1,



a) RED queue at the bottleneck



b) DropTail queue at the bottleneck

Figure 4.24: TCP under the influence of MulTFRC; the bottleneck capacity was changed from 10 Mbit/s to 70 Mbit/s and parameter $n = 1..30$, RED and DropTail queue on the bottleneck link

meaning it obtains its fair share of the bandwidth. The utilization of the bottleneck link was above 93%, and TCP always had more than 90% of his fair share — in most simulations it was around 99%. A lower TCP fair share of the link is not always a sign of MulTFRC being more aggressive since in most of these cases, there was a lower overall link utilization. The difference between the throughput of TCP and MulTFRC was almost always smaller than 5% of the throughput of the TCP flows. It only exceeded 10% of the TCP flows' throughput for some simulations with the 10 Mbps bottleneck link. For this bottleneck capacity, TCP or MulTFRC did not always have an advantage but the situation changed depending on the value of parameter 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. TCP had a higher rate in simulations where a high loss was seen (in this figure, this is for the bottleneck capacity of 10 Mbps, 30 TCP flows and MulTFRC with $n = 30$); TCP flows obtained 106% of their fair share and MulTFRC had only around 82%. For this simulation, the loss rate was above 14%. The same mismatch is seen in figure 4.23 for high loss rates (corresponding with large values of n).

We ran similar simulations with a DropTail queue on the bottleneck link. The bottleneck link delay was set to 20 ms and the capacity was varied from 10 Mbit/s to 70 Mbit/s (the same as in the previous simulation set). The queue parameters were the same as in [50]. The buffer of the DropTail queue was equal to three $\times$ bandwidth $\times$ delay. To avoid phase effects, we changed the delay on the right-hand side; for TCP flows it had different values between 0 and 2 ms. As Figure 4.24 (b) shows, MulTFRC does not significantly affect the throughput of TCP flows, and in cases of greater mismatching, TCP almost always obtained a higher rate (only for a capacity of 50 Mbps and n equal to 2, MulTFRC had a significantly higher rate). The high link utilization was seen in this simulations as well (always above 97%; mostly larger than 99%).

MulTFRC almost always had more than 90% of its fair share of the bandwidth. For the bottleneck capacity of 10 Mbps 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 eliminated completely. For the other values of the capacity, MulTFRC obtained a more than 7% lower rate than the TCP flows only when there was a very low loss rate which is due to an error introduced by the model (the reader is referred to the discussion about Figures 4.3 and 4.4). The use of the improved j calculation shows an advantage. As the parameter n was set to a number smaller than 13, MulTFRC obtained such a lower rate only for loss rates smaller than 0.1%; for the other values of n this error appeared for loss rates smaller than 1%. It would be interesting to further study a possibility of using the improved j also for $n > 12$ in case a very low loss rate is experienced. We leave such a study as a possible future work.

We also study the behavior of MulTFRC for a large value of n. Figures 4.25 and 4.26 show results with a bottleneck capacity of 64 Mbps and 128 Mbps. In Figure 4.25, the largest difference between MulTFRC and TCP flows is for n equal to 15; in this case, MulTFRC has 8% lower rate than the TCP flows. For other values of n, the difference does not exceed 5% of the TCP flows' throughput, and even a very small difference of 0.5% can be seen (e.g. for n equal to 75 and 80).

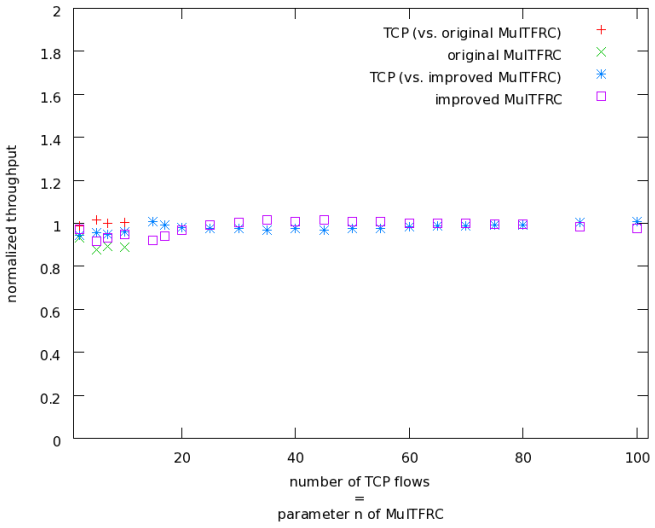


Figure 4.25: TCP-friendliness of MulTFRC with old and new calculation of j (which only differs for $n < 13$): the bottleneck link capacity was 64 Mbps and the link delay was 20 ms; all values are between 0.89 and 1.01

The results with a bottleneck capacity of 128 Mbps are shown in Figure 4.26. MulTFRC has a 10% lower rate than TCP flows only for n equal to 15, 17, 20 and 25. For these values of n, the loss rate was smaller than 0.5%, and this mismatch can be due to an error introduced by the equation. As already mentioned, the equation gives a low rate for very small loss rates (the reader is referred to Section 3.3.4 for more details). An even smaller loss rate was seen for n smaller than 15, but in these cases, the use of the improved j calculation enabled MulTFRC to compete better with TCP flows. Also here, the use of the improved j calculation for parameter n equal to 15, 17, 20 and 25 and a low loss rate could be investigated.

We also study the MulTFRC and TCP behavior in the presence of web-like background traffic. We want to investigate how MulTFRC influences TCP flows by comparing the throughput of TCP flows when they are sharing the link with MulTFRC and the throughput of TCP flows when instead of MulTFRC flows a corresponding number of TCP flows are present. We ran two sets of simulations. The first set had the same simulation setup as in the previously shown simulation, but web traffic was introduced. To remind the reader, there were n TCP flows and MulTFRC had the aggression parameter equal to n. In the second set of simulations, n TCP flows shared the link with n other TCP flows instead of MulTFRC (there are $2n$ flows and only the first n TCP flows are of interest, we will call them the 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 the ns-2 simulator [26]. The same normalization was used as in Figure 4.25, only here the available bottleneck bandwidth was decreased by the bandwidth which is occupied by the background traffic and

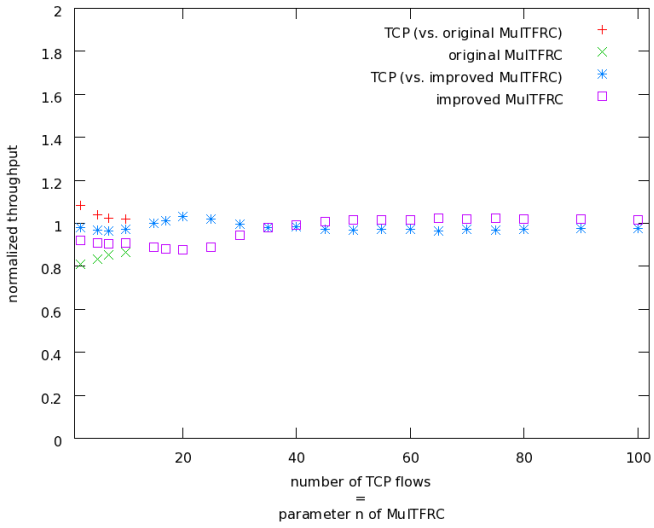


Figure 4.26: TCP-friendliness of MulTFRC with old and new calculation of j (the difference is only for $n < 13$): the bottleneck link capacity was 128 Mbps and the link delay was 20 ms; all values are between 0.8 and 1.08

therefore also here, an ideal normalized throughput should be 1. In Figure 4.27, we compared the normalized throughput of the observed TCP flows in the simulations with MulTFRC and the observed TCP flows competing with TCP. MulTFRC with the improved j calculation was used. MulTFRC occupied the remaining bandwidth and therefore almost always achieved the normalized throughput closed to 1. For 10% background traffic, the difference between the throughput of TCP flows competing with MulTFRC and the throughput of TCP flows competing with TCP flows is slightly higher for n equal to 2; in this case, TCP competing with MulTFRC achieves an 8% higher rate than when it was competing with TCP. This means that MulTFRC does not harm TCP flows — on the contrary, it achieved a slightly lower rate than its fair share of the bandwidth (it achieved only 84%; as already stated, such a result is due to an error introduced by the equation). For other values of n, the throughput of TCP flows competing with MulTFRC deviates from the throughput of TCP flows competing with TCP flows by less than 3.5% and in most cases, the difference is around 1%. In the range of n from 5 to 10 and between 17 and 60, TCP competing against MulTFRC achieves a smaller throughput than when it was competing with TCP flows; in the rest of the simulation, it is the opposite case. As this difference is small, it means that MulTFRC does not harm TCP traffic. The difference between MulTFRC and TCP was between 4.3% and 0.6% of the TCP throughput which means that MulTFRC achieves a similar rate as TCP flows even in the presence of web-like background traffic.

As the background traffic occupied 45% of the link, the results slightly changed. For $n = 2$, TCP competing against MulTFRC achieves a 5.9% higher rate than the TCP rate when it was

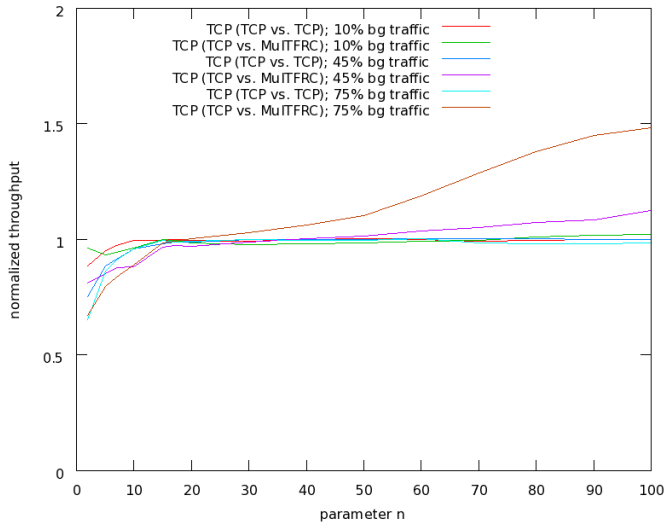


Figure 4.27: TCP-friendliness of MulTFRC: the normalized throughput of n TCP flows sharing the link with a MulTFRC flow (the aggression parameter was the same as the number of the TCP flows) was compared with the normalized throughput of n TCP flows sharing the link with n TCP flows; the bottleneck link capacity was 64 Mbps and the link delay was 20 ms; 10%, 45% and 75% web-like background traffic

sharing the link with other TCP flows. In the range from $n = 2$ to $n = 30$ it has a lower throughput (it has a between 3% and 1.1% lower throughput than the TCP throughput when it was sharing the link with the TCP flows; this value decreases as n grows). For $n > 30$, TCP competing against MulTFRC achieves a higher throughput and the difference grows as n increases; it has a between 0.5% and 12% higher rate than TCP competing with the other TCP flows. The difference is higher than 10% only for $n = 100$. This is due to a high loss rate of more than 11% in which case MulTFRC obtains less than its fair share of the bandwidth. A similar effect can be seen for the background traffic of 75%. Here, already with $n = 50$, TCP competing against MulTFRC achieves a 10% higher rate than when it was competing with the TCP flows. In the range of n between 2 and 40, the difference between the throughput of TCP competing against MulTFRC and the throughput of TCP competing against TCP flows is between 0.1% and 7% of the TCP throughput when it was competing with TCP.

The aggression parameter $n \in \mathbb{R}^+$

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 bottleneck link, which had a delay of 20 ms; the same parameters for the RED algorithm as in the previous subsection

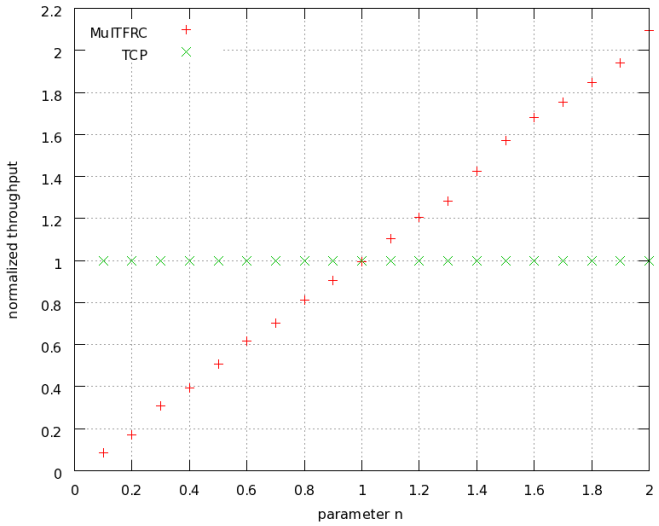


Figure 4.28: A single TCP flow and a MulTFRC flow with n changing from 0 to 2; bottleneck link capacity 8 Mbit/s

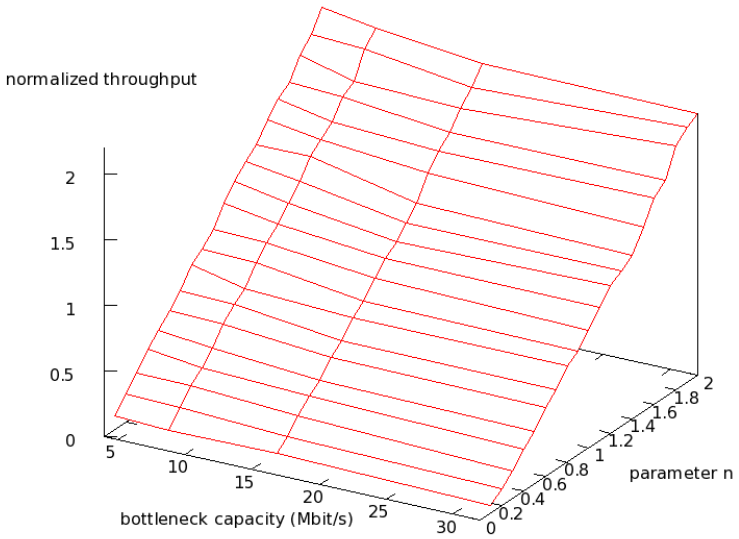


Figure 4.29: A single TCP flow and a MulTFRC flow with n changing from 0 to 2 with a varying bottleneck link capacity (from 4 Mbit/s to 32 Mbit/s)

were used. The displayed values are normalized against the throughput of the TCP flow, and the normalization implies the throughput of the TCP flow to be always equal to one, and the normalized throughput of MulTFRC should ideally be equal to n. The simulations lasted 200 seconds and the last 185 seconds 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 is shown here.

Figure 4.28 shows results with a bottleneck capacity of 8 Mbit/s. We will observe the difference between the normalized throughput of MulTFRC and its fair share (this is equal to parameter n). If we look at the average values, the difference between MulTFRC and its fair share is never greater than 4% of the fair share except for $n = 0.1$ and $n = 0.2$; there the difference is around 15%. By observing each simulation separately and not only the average value, the difference was greater than 10% of the fair share in more than 2 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, as will be explained later, in such cases, MulTFRC sometimes obtains a lower fair share of the bottleneck capacity). The standard deviation of the difference between MulTFRC and its fair shares was mostly smaller than 4% of the fair share. This value is higher than 4% only for n equal to 0.1, 0.2 and 0.3.

For further evaluation with a broader range of network conditions, we also varied the bottleneck capacity, as shown in Figure 4.29. The bottleneck capacity had values between 4 Mbit/s and 32 Mbit/s (x-axis) and n changed from 0.1 to 2.0 (y-axis). Similar results are seen as in Figure 4.28. The difference between the throughput of MulTFRC flow and its fair share is almost always less than 10% of the fair share. Only for a very small loss rate, this difference is higher. This is seen for a loss rate smaller than 0.1% (for example, as in the simulations with 32 Mbps and 16 Mbps bottleneck link and n smaller than 0.7 and 1.4, respectively). This can be due to the error introduced by the equation. As seen in Figures 4.3 and 4.4, for a loss rate of 0.1%, the model deviates from the measured throughput by about 4%. The standard deviation of the difference between TCP flow and MulTFRC flow was almost always less than 6% of the fair share for $n > 1$ and it reached values close to 10% only for n =0.1, 0.2 and 0.3.

4.2.4 Comparison with related work

In Section 2.5, at the beginning of this thesis, an overview of the proposed protocols with a behavior that does not match TCP was given. As already stated there, the protocols that provide a weighted aggression (e.g., [36, 77, 59, 101]) are of special interest to us, since they provide the same service as the protocol presented here. Hence a deeper investigation of these protocols is necessary.

As already mentioned, the Coordination Protocol (CP) is the most similar protocol to the one presented in this thesis. For completeness and better understanding of dissimilarities between these two protocols as well as the performance difference, some of the aspects of the CP protocol, in short presented in Section 2.5, will be repeated here. 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 [103] with n in the protocol. As the authors of [101] have shown, and as we have argued in Section 3.1, 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 this protocol would not work for $0 < n < 1$.

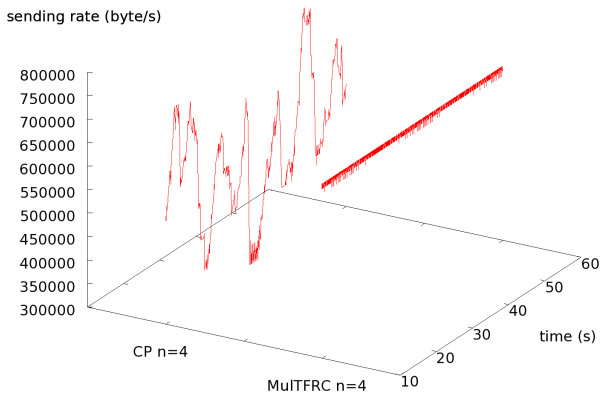


Figure 4.30: Smoothness of MulTFRC and CP: 32 Mbps and 20 ms bottleneck link

Because of using a stochastic technique, this protocol fluctuates more than ours. This is shown in Figure 4.30, which was generated using the original code that was provided to us by the authors of CP[1]. We used the same setup as the one that we used for testing smoothness in Section 4.1; MulTFRC with $n = 4$ and CP with 4 flow shares traversed the same bottleneck link with periodic loss (MulTFRC and CP were run in separate simulations).

Since CP is based on simply 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 3.3, our protocol works even better with an increasing number of flows. Figure 4.31 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 Figure 4.23 (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.

Further, two other protocols were tested: MulTCP and Stochastic TCP. We did not run simulations with PA-MulTCP because the original code was not available.

We tested MulTCP [36] in the same simulation setup as CP, using an update that we made to the code that is available under "contributed code" via the main ns-2 webpage to make it work with the most recent version of ns-2. As stated in [36] and as our comparison shows,

[1]We would like to thank David E. Ott and Ketan Mayer-Patel for providing us with their simulation code.

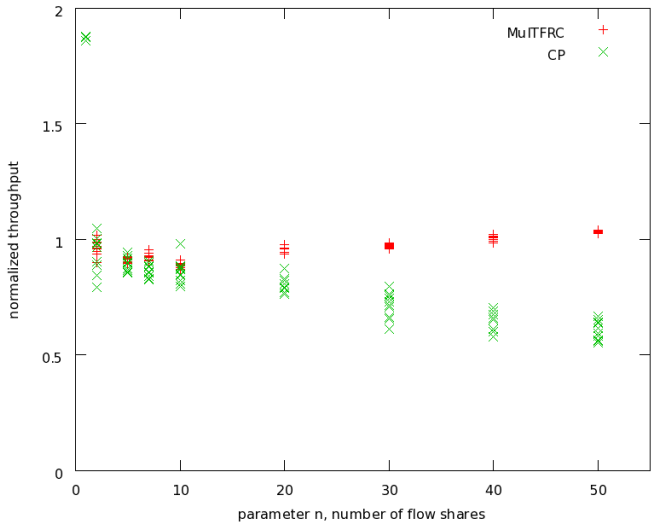


Figure 4.31: MulTFRC vs. TCP and CP vs. TCP with a larger number of flow shares: the bottleneck link capacity was 32 Mbps and the link delay was 20 ms; each simulation was run 10 times and all results are shown

MulTCP performs reasonably well with values up to $n = 10$. This is shown in Figure 4.32. 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 receiver irrespective of their sequence number). This figure was generated using the same simulation setup as for Figure 4.31, we are just zooming into the relevant range. MulTCP shows some deficiencies. Firstly, 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 which yield a satisfactory behavior are quite limited, and choosing $0 < n < 1$ is not possible. Secondly, like standard TCP, MulTCP is not suitable for certain multimedia applications because of its severe rate fluctuations (and extra delay is added to transfers by retransmitting packets). This is, however, not a goal of this protocol.

The figure also shows Stochastic TCP [59], which is another protocol that we tested using the code provided by the authors[2]. 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 Figure 4.32, simulations with Stochastic TCP were run several times).

The name MulTFRC was used before in [31], [32] and [33]. It describes a system that uses a number of separate TFRC connections to improve the performance of TFRC over wireless links. We decided to use it again because of its illustrative nature.

[2]We would also like to thank Thomas J. Hacker for providing us with his simulation code

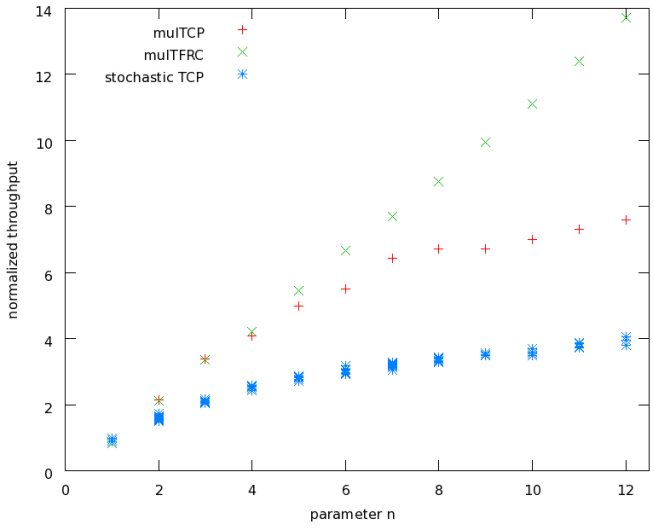


Figure 4.32: MulTFRC, MulTCP and Stochastic TCP: the bottleneck link capacity was 32 Mbps and the link delay was 20 ms

4.3 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[3]. It uses UDP as a transport layer protocol and the rate control according to the MulTFRC protocol is implemented in the application layer. A minor mismatch between the simulation results and real-life measurements can be produced due to additional delay introduced on the hosts by this not-so-ideal implementation. The main changes of the original code are related to accounting for the real loss rate calculation and the equation change. These are the same updates that would be necessary for an extension of a kernel implementation of TFRC (e.g., implementations for Linux and FreeBSD are available[4]). The Internet-Draft extending the specification of the TFRC [51] is given in Appendix B.

Since TFRC was developed with the main idea of providing a congestion controlled service for multimedia applications, the TFRC real-life code implements just an unreliable transfer of data. Because of the weighted congestion control enabled by the MulTFRC which can make it attractive for other applications too, for MulTFRC, in addition to the unreliable variant, a reliable implementation was also developed in a master thesis [113] under the authors supervision.

[3] http://www.icir.org/tfrc/code
[4] http://www.read.cs.ucla.edu/dccp/#implementations

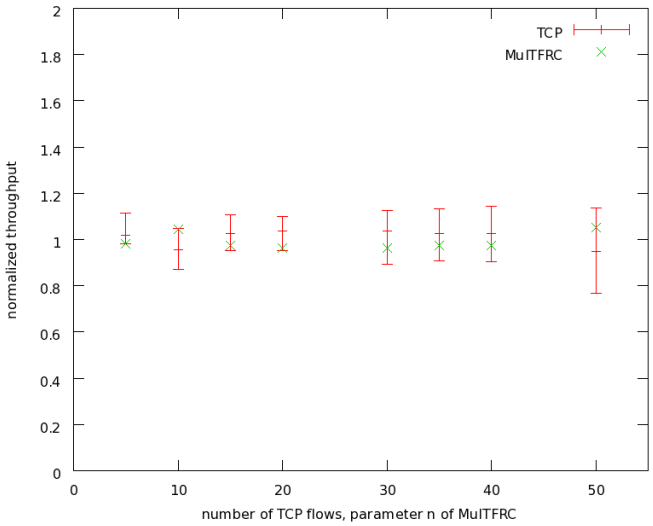


Figure 4.33: Real-life tests in local testbed: MulTFRC vs. TCP

The reliable implementation is available from the MulTFRC project website[5] in the form of a simple tool for transferring files between two hosts. Reliability is ensured in a "selective-repeat" form.

Tests in a local Testbed

We tested MulTFRC against standard TCP with a real-life implementation. The first tests were run in a local testbed. It consists of three hosts, one of which is acting as a router and the other two are the sender and the receiver for TCP flows as well as for MulTFRC. All hosts run the Linux kernel version 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 Mbit/s. TCP flows and a MulTFRC flow were started at the same time. The number of TCP connections was the same as the parameter n of the MulTFRC flow. The TCP Sack was used. The packet size was 1448 bytes for TCP and 1460 bytes for MulTFRC (the header size is not included). The throughput was measured as the number of packets sent per second.

For each measurement, the throughput of the TCP flows and the MulTFRC flow was normalized — each throughput value 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 average throughput was calculated by: $B_{avr} = (\sum_{i=1}^{n} B_{TCP_i} + B_{MulTFRC})/(2n)$ where B_{TCP_i} is the throughput of the i-th TCP flow and $B_{MulTFRC}$ is the throughput of the MulTFRC flow. The throughput of the MulTFRC flow was also divided by n. Figure 4.33 shows that the real-life

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

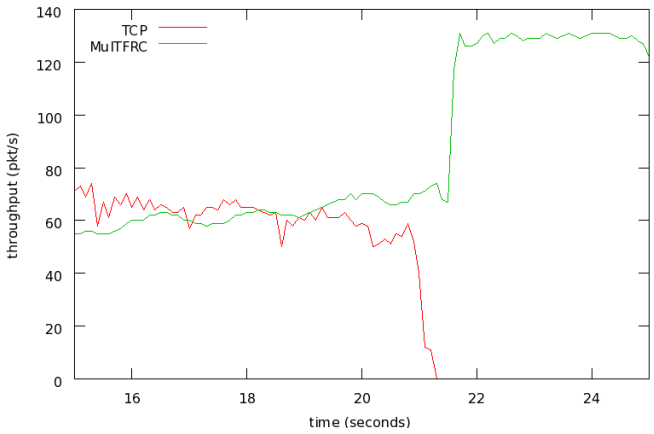


Figure 4.34: Real-life tests in local testbed: MulTFRC responsiveness on TCP traffic; a single MulTFRC flow with parameter n set to 4 and 4 TCP flows; the TCP flows terminate at the 21st second

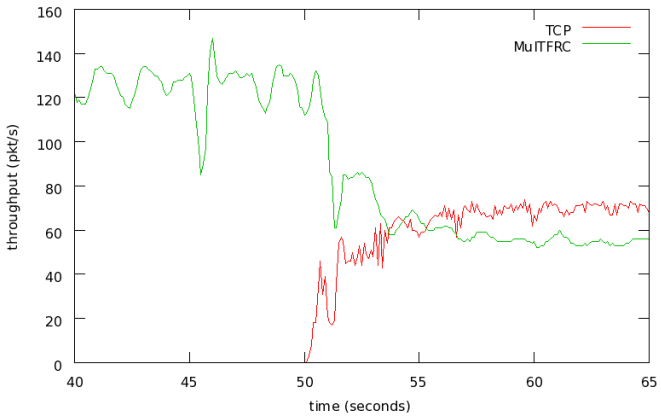


Figure 4.35: Real-life tests in local testbed: MulTFRC responsiveness on TCP traffic; a single MulTFRC flow with parameter n set to 4 and 4 TCP flows; the TCP flows start at the 50th second

implementation yields quite similar results as the ns-2 simulations shown in Figure 4.23. The difference between the normalized throughput of TCP and MulTFRC is always less than 0.1 and it is highest for n equal to 10 and 50. The same can be seen in the simulations, but the difference in the simulations is slightly smaller. In Figure 4.33 we also show the normalized minimal and maximal throughput obtained by individual TCP flows.

In subsection 4.2.2 we studied the responsiveness of MulTFRC using simulations, now we will investigate how the MulTFRC real-life implementation behaves in the presence of TCP traffic. We ran two tests. In all tests MulTFRC had the parameter n set to 4 and there were 4 TCP flows present in the network. The bottleneck link had a delay of 20 ms and a capacity of 15 Mbps (the *tc* command was used). This is the same setting as in the simulations shown in Figure 4.21. In the first scenario, the TCP flows and a MulTFRC flow started at the same time. After they had reached stable rates, the TCP flows terminated. This is shown in Figure 4.34. As it can be seen, the MulTFRC flow reacts fast and quickly utilizes this new available bandwidth. Figure 4.35 shows the opposite scenario. In the first 50 seconds, MulTFRC is the only flow in the network, and then 4 TCP flows start. Since the testbed is rather small, a route change could not be performed and the TCP flows were not having the steady-state behavior but they were in the slow start phase at the time they interacted with MulTFRC. As it can be seen in Figure 4.35 MulTFRC reacts faster than shown in the simulation (Figure 4.21).

Tests in PlanetLab

To validate MulTFRC on the Internet, we used PlanetLab [2]. Since our implementation of MulTFRC uses UDP as the transport layer protocol and since 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 MulTFRC code is based on the TFRC code, which is a user space implementation and therefore can not make use of TCP packets (packets with a TCP protocol number) to avoid the rate limit problem. We tried to use the same hosts as for the model validation, presented in subsection 3.3.5. Only with two of these hosts the testing was possible; in other cases, an even more than 6 times greater loss rate of UDP packets than the loss rate of TCP packets was seen.

The measurements were obtained on the 15th of October 2009 and between the 13th of January 2010 and the 15th of January 2010. MulTFRC with parameter n started at the same time as n TCP flows and their average throughput over 200 seconds was measured. As in all previous measurements the start-up phase of both protocols was not taken into account for the measurements (the first 60 seconds were neglected). TCP flows had window scaling turned on and the advertised window scaling value was 7. TCP flows were neither sender nor receiver limited. The parameter n took values 10, 15 and 20. The same normalization and the same packet sizes as with the local testbed measurements were 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 host at Universite De Technologie De Troyes, Troyes, France (planetlab2.utt.fr — 194.254.215.12). This host ran PlanetLab software (kernel version $2.6.22.19 - vs2.3.0.34.39.planetlab$). The corresponding results for n equal to 10, 15 and 20 are

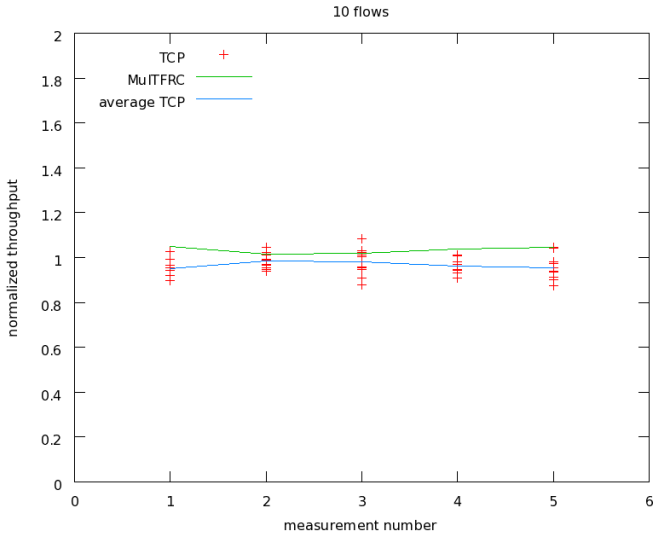


Figure 4.36: MulTFRC on PlanetLab — France; 10 TCP flows vs. MulTFRC with n=10

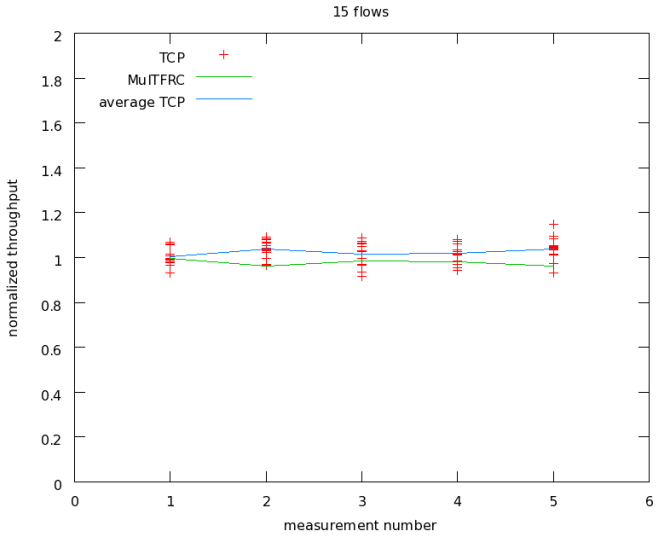


Figure 4.37: MulTFRC on PlanetLab — France; 15 TCP flows vs. MulTFRC with n=15

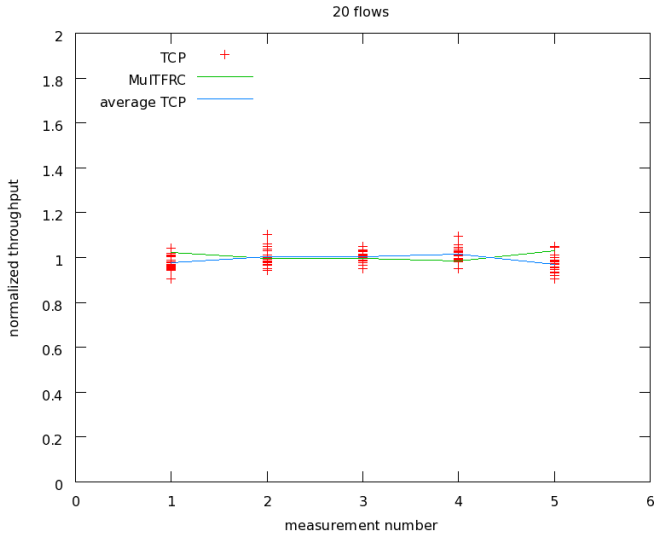


Figure 4.38: MulTFRC on PlanetLab — France; 20 TCP flows vs. MulTFRC with n=20

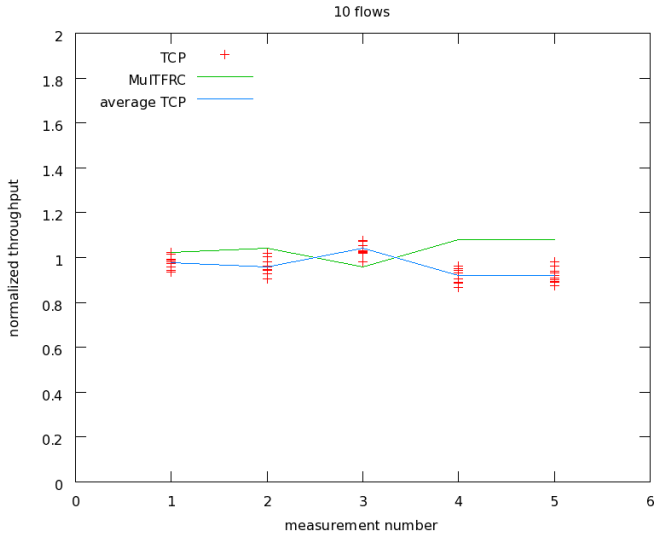


Figure 4.39: MulTFRC on PlanetLab — Portugal: 10 TCP flows vs. MulTFRC with n=10

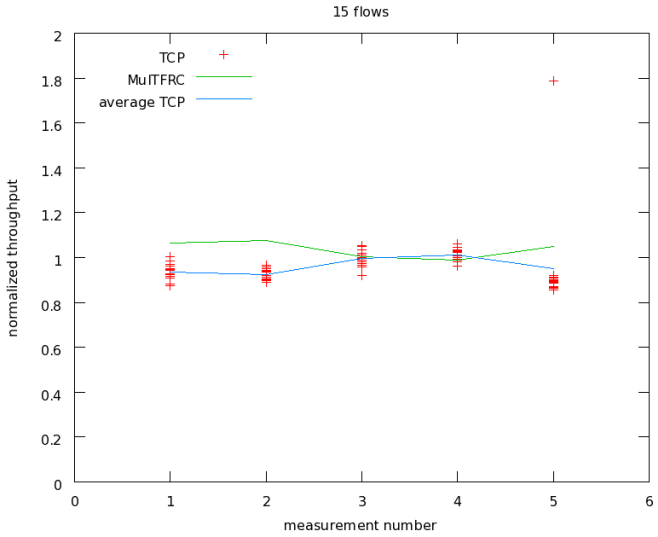


Figure 4.40: MulTFRC on PlanetLab — Portugal: 15 TCP flows vs. MulTFRC with n=15

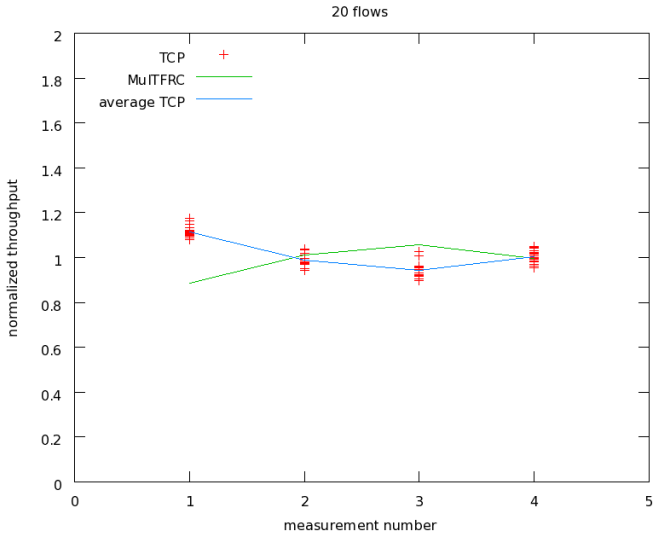


Figure 4.41: MulTFRC on PlanetLab — Portugal; 20 TCP flows vs. MulTFRC with n=20

shown in Figures 4.36, 4.37 and 4.38. The results are very good, as the difference never exceeds 10% of the TCP throughput. Separate measurements gave slightly different results. For n equal to 10, MulTFRC has a between 3.2% and 9.7% higher rate than TCP; also in the simulations (Figures 4.23, 4.25 and 4.26), MulTFRC had a higher rate for n equal to 10. Figure 4.37 shows that MulTFRC has a between 0.8% and 7.8% lower throughput than the TCP flows for n equal to 15. The results for n equal to 20 are very good too; some tests showed a difference between TCP flows and a MulTFRC flow of only 0.39% and 0.66% of the TCP throughput. The highest difference seen for n equal to 20 (in Figure 4.38) is 6% of the TCP flows' throughput.

The second PlanetLab host was situated at Universidade do Algarve, Algarve, Portugal (planetlab1.fct.ualg.pt — 193.136.227.163). This host ran PlanetLab software (kernel version 2.6.22.19−*vs*2.3.0.34.39.*planetlab*). On our site in Innsbruck, the same host was used as in the previously presented measurements.

Figures 4.39, 4.40 and 4.41 show results of these measurements. Here for some measurements a greater difference can be seen. For n equal to 10 (Figure 4.39), there is a range from a small difference of only 4% of the TCP throughput to a high difference of 16%. In most cases, MulTFRC has a higher rate than TCP; the opposite is seen in only one case where MulTFRC has an 8% smaller rate. Similar results are seen for n equal to 15 and 20; for $n = 15$, MulTFRC has a between 0.9% and 15% higher rate than the TCP flows. For n equal to 20, only four measurements were performed because the host became unavailable. In two measurements with n equal to 20, MulTFRC had a higher rate (it had a 2% and 11% higher rate than TCP), and the other two times TCP flows had a 1% and 22% higher rate than MulTFRC.

4.4 Application

A congestion control mechanism which is "n-TCP-friendly" can yield an immediate practical benefit for individual users in the Internet of today, especially if it allows a broad range of values for a flow's aggression (n), as MulTRFC does. MulTFRC also shows a desirable behavior in case of n being a positive real number between zero and one, by offering a "less-than-best effort" as well as a "better-than best effort" service in this way. Hence, there are several perspectives of the protocol usage depending on the user's interest.

As many Internet users know, multiple TCP connections are faster than a single one. Examples are peer-to-peer applications which use multiple connections and GridFTP [7] — an extension of FTP which utilizes multiple TCP connections. MulTFRC provides the same service by applying the aggression parameter $n > 1$ and, as shown, gives good results even for a value larger than 100. Opening multiple standard TCP connections introduces a certain overhead, as the state information must be kept for each flow and data transfers must be multiplexed onto individual TCP flows (e.g., by offering large files which are split into several parts for easier parallel downloading). On the other hand, MulTFRC uses just one connection and therefore the above stated disadvantages do not apply for it.

As stated in the discussion about fairness in Section 2.6, the current services offered in the Internet cannot satisfy the user perspectives of importance. TCP is currently almost the only reliable transport layer protocol used in the Internet, and it provides an equal share of the bandwidth between all flows. From the user's point of view, the benefit of all flows is not always the same, e.g., a user can be more eager to obtain file A than file B which are transferred simultaneously, and there can be situations where a peer-to-peer application could disturb a streaming video. In a common home network setup, where the narrow-band access link (e.g., dial-up, DSL, etc.) is usually the bottleneck, the user's own flows are actually only competing with each other and not with "foreign" traffic. In such a situation, sharing the available bandwidth according to the user's needs is the only fairness notion that truly counts. While TCP-friendliness could theoretically be neglected in such a scenario, e.g., by controlling the sending rate with a simple queuing scheme at the access router, doing so would, however, be harmful when the bottleneck shifts, e.g. when congestion happens in other parts of the network. In such a case, differentiated fairness on the bottleneck is about users competing with each other, where some users might be willing to pay more for getting a better quality of service. MulTFRC can be a solution to the above stated problem, by simply assigning proper weights (the aggression parameter n) for each flow.

A similar problem can be found in the case of a distributed application; there may be parallel flows of different importance which belong to the same user (e.g., a data transfer which is an integral part of a workflow, and a data transfer for replication in a computational Grid). Then, it could once again be useful to have a protocol like MulTFRC.

One case of particular interest is $n < 1$; an n-TCP-friendly protocol with such a parameter choice could possibly satisfy a demand that was raised at the 72nd IETF meeting in Dublin, Ireland. At this meeting, a "BOF" session was held under the heading of "Techniques for Advanced Networking Applications (TANA)", where the main focus was on transport layer support for P2P applications. Then an IETF working group "Low Extra Delay Background Transport" (LEDBAT) was formed. There, developing a congestion control mechanism for background traffic that minimizes the delay that it introduces into the network is set as the main goal. The intention is, for example, to prevent a P2P application from causing harm to the communication of an interactive online game across a thin uplink home connection. It seems that MulTCP could be a suitable solution for all these scenarios, although this working group has chosen to work in the direction of a delay based congestion control algorithm.

At the 75th IETF meeting in Stockholm, Sweden, a suggestion for standardization of a multipath variant of TCP was proposed, and the formation of a working group was approved. Multipath TCP should utilize parallel paths and at the same time ensure that, if the paths share the same bottleneck link, the aggregate flow is not more aggressive than a single conforming TCP flow, i.e. that the aggregate flow is "TCP-friendly". At this meeting a solution was presented that achieves such a behavior only by changing the congestion control algorithm without additional tools for shared-bottleneck discovery. Similar proposals for parallel flows are presented in [65, 58]. In both cases each flow is less aggressive than a standard TCP flow and the aggression of the aggregate flow is comparable to that of a standard TCP flow. In [65] this is achieved by changing the increase parameter of TCP congestion control and in the second case [58] a longer "virtual" round-trip time is used. We are tempted to believe that the

same behavior for multimedia application can be achieved with MulTFRC, using an additional technique for distribution of the aggression parameter between paths under the constraint that the sum of the parameter values per path does not exceed one.

On a side note, in [101], CP is described as a means to efficiently transfer aggregates of multimedia streams between edge devices (so-called "aggregation points") in the Internet — this is yet another way of using the MulTFRC protocol.

Here we gave just some hints of possible applications of the MulTFRC protocol. Whether there are further examples of MulTFRC's usefulness is to be seen.

Chapter 5

Conclusion

The TCP congestion control mechanism made the Internet stable and has managed to keep it so over the years, but the use of the TCP protocol and "TCP-friendly" protocols does not seem to always satisfy users' needs. Since several TCP flows are more aggressive than a single one, the use of parallel flows to saturate high-speed links or to differentiate between flows has become common. This thesis has focused on parallel TCP flows and the main contributions are 1) two practical models of parallel TCP flows and 2) MulTFRC, a protocol that behaves like a number of parallel TCP flows.

In the first part of this thesis we concentrated on the modeling of the steady-state throughput of parallel flows depending on the network conditions observed by a TCP sender. The network is characterized by the packet loss rate and the round-trip time seen by the senders. As parallel flows can be observed individually or as an aggregate, the outcome are two equations — one for each case. Both equations are very accurate and easy to use at the same time.

Observing packet loss on a per-flow basis (observing each flow individually) yielded an easier final equation, although measuring loss in this way is more demanding since accounting for each flow must be done separately. On the other hand, an extension of this equation, assuming only the loss event rate of the cumulative flow is known, enables an easier loss estimation without increasing the complexity of the calculation. An extended evaluation study, using simulations and real-life measurements, showed a satisfying accuracy of both models over a broad range of network conditions. These results also scale well with the number of observed flows. We therefore believe that these equations can have a wide range of possible applications. An application of these equations is the second outcome of this thesis — the MulTFRC protocol. Other straightforward ideas on how these equations could be used were discussed, e.g., predicting the throughput and the latency of parallel transfers.

Although the models are already very accurate in a broad range of network conditions it would be interesting to further study parallel TCP flows by relaxing some of the assumptions, e.g.,

assumptions about independence of the round-trip time and the window size. Further, different distributions of losses in a loss event of the cumulative flow could be investigated, e.g., any distribution in the range of two extremes: completely correlated loss (each loss event affects all flows) and the opposite, in each loss event of the cumulative flow just one flow experiences loss. Incorporating the variation of the round-trip time, as seen in our real-life tests, is also an interesting topic for future work; how this variation influences the relationship between the average round-trip time and the average duration of a round (the time between two consecutive window increases) could be further studied. These models do not incorporate a possibility of the window being limited by the receiver. However, since growing use of the window scaling option can be seen, we believe that the modeling of this aspect is not crucial.

MulTFRC is a rate-based protocol that uses our model with the loss measure of the cumulative flow. It calculates the rate depending on the network conditions and the chosen aggression parameter — n. In this way, the "n-TCP-friendliness" of the protocol is accomplished. MulTFRC provides smoother rates of its flows which makes it suitable for real-time applications, but the flexibility that MulTFRC offers makes it attractive for other transfers too. A simple change of the weight regulates the aggression of a connection. Such a service could be useful for a differentiation of connections by allowing users to choose the weight of a transfer depending on their benefit from the transfer. Considering that the MulTFRC protocol also performs well for values of the aggression parameter n being less than one, a "less-than-best effort" service for background traffic can also be achieved using MulTFRC.

The performance of the MulTFRC protocol was evaluated using simulations (using the ns2 simulator) and real-life tests. Both showed that MulTFRC behaved as expected for a broad range of values for the aggression parameter n (for $0 < n \leq 100$) and varying network conditions. We also provided the Internet-Draft specification for MulTFRC (Appendix B).

TFRC extensions were proposed for some special use cases, e.g. the small-packet (SP) variant of TFRC [54], TCP-Friendly Multicast Congestion Control (TFMCC) [127] and an extension of TFRC for a faster restart. As we will discuss in the following, such extensions could be also considered for MulTFRC.

Some applications are designed to send small packets or to have a certain fixed rate, but change the packet size depending on network conditions. Such applications sending over the standard TFRC would receive a smaller rate in bytes pre second than TCP with a larger packet size. The authors of [54] and [126] suggest a small-packet variant of TFRC which should obtain the same rate in bytes per second as a TCP flow using packets of 1460 bytes. This is achieved by using a segment size of 1460 bytes for the rate calculation. But a problem arises from the fact that a flow sending a larger number of small packets receives a smaller loss rate than a flow with a larger packet size, and it therefore obtains a higher rate [126]. The difference in the seen loss rate depends on the mode in which queues operate; queues can drop packets independently of the packet size (the packet mode) or they can operate in the byte mode, and then the dropping probability depends on the packet size. The authors of [126] propose two separate mechanisms for the two queue modes for improving the loss rate measurement and thereby improving the fairness of the small-packet variant of TFRC towards TCP. Choosing a mechanism is not a trivial task. The solution proposed in [54] performs fairly well in both

cases, although in the byte mode case, it can receive a lower loss rate and higher throughput. To avoid such an unfair behavior, the authors of [54] proposed using the small-packet variant just for applications sending not more than one packet every 10ms. Further, for loss event intervals shorter than two round-trip times, the use of the loss rate instead of the loss event rate was also suggested. To prevent applications from obtaining a higher rate by using smaller packets, accounting for the bandwidth used for the packet header is obligatory.

Since a MulTFRC flow with parameter n receives an n times larger rate than a standard TCP flow, MulTFRC could probably directly be used as a small-packet variant of TFRC. With a queue operating in packet mode, simply setting n to $packet\_size/1460$ ($packet\_size$ is the size of a packet in bytes used by the small-packet variant) could enable the MulTFRC small-packet variant to receive approximately the same rate as a conforming TCP flow using packets of size 1460 bytes. This small-packet variant could be slightly more aggressive than TCP with queues operating in byte mode. The loss event rate would be the same but it is to be expected that the number of flows reducing their rate in a loss event of the cumulative flow would be small and the MulTFRC small-packet variant would always reduce its rate for a factor smaller than 1/2. As the loss rate grows, it is to be expected that the MulTFRC small-packet variant would come closer to the rate of a TCP flow. But this certainly needs a deeper investigation.

TCP-Friendly Multicast Congestion Control (TFMCC) is a variant of TFRC designed for multicast applications. A simple use of MulTFRC in such a case is possible. But some questions arise: How a value of the parameter n could be distributed? Is it possible that receivers use different values with receivers experiencing higher loss rate chose a higher value of n? Such an extension of MulTFRC could be a possible item of future work.

Faster restart of TFRC is suggested to improve the performance of interactive applications [74]. It could also be used with the small packet variant of TFRC. Interactive applications switch between active parties and usually just one party is active at a time. During an idle time period, TFRC reduces its sending rate by half each time the “nofeedback” timer expires (the nofeedback timer is set to 4RTT). This causes TFRC to enter the slow start phase after such a period. This is a very conservative behavior and it degrades the performance of interactive applications significantly. In [74], faster restart after an idle period is proposed. If a sender has been idle for less than 10 minutes, it is allowed to start sending at the same rate as before the idle period. The allowed rate is further linearly decreased to zero as the idle period increases to 30 minutes. The same feature could be applied for MulTFRC. Since MulTFRC acts like a number of flows.

The study documented in [111] states that there can be an imbalance in the long-term throughput of TFRC and TCP. This is due to the fact that TFRC encounters a loss rate which is lower than the one observed by a standard TCP flow. Here this issue is not studied and it is to be expected that MulTFRC will show the same behavior.

Newly started activities to obtain a TCP variant for a transfer over multiple paths (MPTCP — multipath TCP) lead to an idea that a similar protocol could be provided for multimedia traffic too; real-time applications would have an even greater benefit from such a service (being able to a split transfer between multiple paths and in some way being able to choose paths with

a lower loss rate). As emphasized in section 4.4, MulTFRC could easily be extended to offer a multipath congestion controlled service for multimedia traffic.

Throughout this thesis we studied parallel TCP flows and we showed that our models as well as MulTFRC show desirable behavior. As our discussions indicate, we believe that the work presented here offers a wide range of possible further use.

Appendix A

Algorithm

In Chapter 4, we improved our model from Section 3.3. Here we incorporate these changes into algorithm 2. The use of this improved algorithm is not limited to MulTFRC only and it could be used instead of algorithm 2 on page 72.

- RTT — is the round-trip time in seconds.
- b — is the maximum number of packets acknowledged by a single TCP acknowledgement.
- p_e — is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.
- j — is the number of packets lost in a loss event.
- t_RTO — is the TCP retransmission timeout value in seconds.
- n is the number of TFRC flows that MulTFRC should emulate. It is a positive rational number.

and x, j_af, a, z and q are temporary floating point variables. The algorithm returns the throughput in "packet per second".

Here we only present the equation in the form of n algorithm and a possible complete implementation is not suggested. Some issues like, for instance, the best way to obtain the RTT measurement, are still left up to the programmer who wants to make use of the algorithm.

Algorithm 3 The throughput of n parallel flows [pkt/s]

if $(N < 12)$ **then**
  $j\_af = N * (1 - (1 - 1/N)^j)$;
else
  $j\_af = j$;
end if
$j\_af = max(min(j\_af, ceil(N)), 1)$;
$a = p\_e * b * j\_af * (24 * n^2 + p\_e * b * af * (n - 2 * j\_af)^2)$;
$x = (j\_af * p\_e * b * (2 * j\_af - n) + sqrt(a))/(6 * n^2 * p)\_e$;
$q = min(2 * j * b/(x * (1 + 3 * n/j)), n)$;
$z = t_R TO * (1 + 32 * p\_e^2)/(1 - p\_e)$;
if $q * z/(x * RTT) \geq n$ **then**
  $q = n$;
else
  $q = q * z/(x * RTT)$;
end if
return ((1-q/n)/(p_e*x*RTT)+q/(z*(1-p_e)));

Appendix B

Internet-Draft

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: April 9, 2010

M. Welzl
University of Oslo
D. Damjanovic
University of Innsbruck
October 6, 2009

MulTFRC: TFRC with weighted fairness
draft-welzl-multfrc-01.txt

Status of this Memo

Copyright Notice

Abstract

This document specifies the MulTFRC congestion control mechanism. MulTFRC is derived from TFRC and can be implemented by carrying out a

few small and simple changes to the original mechanism. Its behavior differs from TFRC in that it emulates a number of TFRC flows with more flexibility than what would be practical or even possible using multiple real TFRC flows. Additionally, MulTFRC better preserves the original features of TFRC than multiple real TFRCs do.

Table of Contents

1. Introduction

"TCP-friendliness", the requirement for a flow to behave under congestion like a flow produced by a conformant TCP (introduced by the name of "TCP-compatibility" in [RFC2309]), has been put into question in recent years (cf. [Bri07]). As an illustrative example, consider the fact that not all data transfers are of equal importance to a user. A user may therefore want to assign different priorities to different flows between two hosts, but TCP(-friendly) congestion control would always let these flows use the same sending rate. Users and their applications are now already bypassing TCP-friendliness in practice: since multiple TCP flows can better saturate a bottleneck than a single one, some applications open multiple connections as a simple workaround. The "GridFTP" [All03] protocol explicitly provides this function as a performance improvement.

Some research efforts were therefore carried out to develop protocols where a weight can directly be applied to the congestion control mechanism, allowing a flow to be as aggressive as a number of parallel TCP flows at the same time. The first, and best known, such protocol is MulTCP [Cro+98], which emulates N TCPs in a rather simple fashion. Improved versions were later published, e.g. Stochastic TCP [Hac+04] and Probe-Aided (PA-)MulTCP [Kuo+08]. These protocols could be called "N-TCP-friendly", i.e. as TCP-friendly as N TCPs.

MulTFRC, defined in this document, does with TFRC [RFC5348] what MulTCP does with TCP. In [Dam+09], it was shown that MulTFRC achieves its goal of emulating N flows better than MulTCP (and improved versions of it) and has a number of other benefits. For instance, MulTFRC with N=2 is more reactive than two real TFRC flows are, and it has a smoother sending rate than two real MulTFRC flows do. Moreover, since it is only one mechanism, a protocol that uses MulTFRC can send a single data stream with the congestion control behavior of multiple data streams without the need to split the data and spread it over separate connections. Depending on the protocol in use, N real TFRC flows can also be expected to have N times the overhead for, e.g., connection setup and teardown, of a MulTFRC flow with the same value of N.

The core idea of TFRC is to achieve TCP-friendliness by explicitly calculating an equation which approximates the steady-state throughput of TCP and sending as much as the calculation says. The core idea of MulTFRC is to replace this equation in TFRC with the algorithm from [Dam+08], which approximates the steady-state throughput of N TCP flows. MulTFRC can be implemented via a few simple changes to the TFRC code. It is therefore defined here by specifying how it differs from the TFRC specification [RFC5348].

2. Specification

This section lists the changes to [RFC5348] that must be applied to turn TFRC into MulTFRC. The small number of changes ensures that many original properties of a single TFRC flow are preserved, which is often the most appropriate choice (e.g. it would probably not make sense for a MulTFRC flow to detect a data-limited interval differently than a single TFRC flow would). It also makes MulTFRC easy to understand and implement. Experiments have shown that these changes are enough to attain the desired effect.

2.1. Section 3 of RFC 5348

While the TCP throughput equation requires the loss event rate, round-trip time and segment size as input, the algorithm to be used for MulTFRC additionally needs the number of packets lost in a loss event. The equation, specified in [RFC5348] as

$$X\_Bps = \frac{s}{R*\sqrt{2*b*p/3} + (t\_RTO * (3*\sqrt{3*b*p/8})*p*(1+32*p^2))}$$

is replaced with the following algorithm, which returns X_Bps, the average transmit rate of N TCPs in bytes per second:

```
If (N < 12) {
    af = N * (1-(1-1/N)^j);
}
Else {
    af = j;
}
af=max(min(af,ceil(N)),1);
a = p*b*af*(24*N^2+p*b*af*(N-2*af)^2);
x= (af*p*b*(2*af-N)+sqrt(a))/(6*N^2*p);
q=min(2*j*b/(x*(1+3*N/j)),N);
z=t_RTO*(1+32*p^2)/(1-p);
If (q*z/(x*R) >= N) {
    q = N;
} Else {
    q = q*z/(x*R);
}
X_Bps = ((1-q/N)/(p*x*R)+q/(z*(1-p)))*s;
```

Where:

s is the segment size in bytes (excluding IP and transport protocol headers).

R is the round-trip time in seconds.

b is the maximum number of packets acknowledged by a single TCP acknowledgement.

p is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.

j is the number of packets lost in a loss event.

t_RTO is the TCP retransmission timeout value in seconds.

N is the number of TFRC flows that MulTFRC should emulate. N is a positive rational number; a discussion of appropriate values for this parameter, and reasons for choosing them, is provided in Section 3.2.

ceil(N) gives the smallest integer greater than or equal to N.

x, af, a, z and q are temporary floating point variables.

Section 3.1 of [RFC5348] contains a recommendation for setting a parameter called t_RTO. Here, the use of the algorithm for calculating RTO specified in [RFC2988] is RECOMMENDED instead. Further, this section proposes a simplification of the equation as a result of setting t_RTO in a specific way. This part of the TFRC specification is irrelevant here. Section 3.1 of [RFC5348] also contains a discussion of the parameter b for delayed acknowledgements and concludes that the use of b=1 is RECOMMENDED. This is also the case for MulTFRC.

Section 3.2.2 of [RFC5348] specifies the contents of feedback packets. In addition to the information listed there, a MulTFRC feedback packet also carries j, the number of packets lost in a loss event.

2.2. Section 4 of RFC 5348

The procedure for updating the allowed sending rate in section 4.3 of [RFC5348] ("action 4") contains the statement:

Calculate X_Bps using the TCP throughput equation.

which is replaced with the statement:

Calculate X_Bps using the algorithm defined in section 3.

2.3. Section 5 of RFC 5348

Section 5.2 of [RFC5348] explains how a lost packet that starts a new loss event should be distinguished from a lost packet that is a part of the previous loss event interval. Here, additionally the number of packets lost in a loss event is counted, and therefore this section is extended with:

If S_new is a part of the current loss interval LP_0 (the number of lost packets in the current interval) is increased by 1. On the other hand, if S_new starts a new loss event, LP_0 is set to 1.

Section 5.4 of [RFC5348] contains the algorithm for calculating the average loss interval that is needed for calculation of the loss event rate, p. MulTFRC also requires the number of lost packets in a loss event, j. In [RFC5348] the calculation of the average loss interval is done using a filter that weights the n most recent loss event intervals, and setting n to 8 is RECOMMENDED. The same algorithm is used here for calculating the average loss interval. For the number of lost packets in a loss event interval, j, the weighted average number of lost packets in the n most recent loss intervals is taken and the same filter is used.

For calculating the average number of packets lost in a loss event interval we use the same loss intervals as for the p calculation. Let LP_0 to LP_k be the number of lost packets in the k most recent loss intervals. The algorithm for calculating I_mean in Section 5.4 of [RFC5348] (page 23) is extended by adding, after the last line ("p = 1 / I_mean;"):

```
LP_tot = 0;
If (I_tot0 > I_tot1) {
    for (i = 0 to k-1) {
          LP_tot = LP_tot + (LP_i * w_i);
    }
}
Else {
    for (i = 1 to k) {
          LP_tot = LP_tot + (LP_i * w_i);
    }
}
j = LP_tot/W_tot;
```

In section 5.5 of [RFC5348] (page 25), the algorithm that ends with "p = min(W_tot0/I_tot0, W_tot1/I_tot1);" is extended by adding:

```
LP_tot = 0;
If (I_tot0 > I_tot1) {
    for (i = 0 to k-1) {
        LP_tot = Lp_tot + (LP_i * w_i * DF_i * DF);
    }
    j = LP_tot/W_tot0;
}
Else {
    for (i = 1 to k) {
        LP_tot = LP_tot + (LP_i * w_(i-1) * DF_i);
    }
    j = LP_tot/W_tot1;
}
```

2.4. Section 6 of RFC 5348

The steps to be carried out by the receiver when a packet is received in section 6.1 of [RFC5348] ("action 4") contain the statement:

3) Calculate p: Let the previous value of p be p_prev. Calculate the new value of p as described in Section 5.

which is replaced with the statement:

3) Calculate p and j: Let the previous values of p and j be p_prev and j_prev. Calculate the new values of p and j as described in Section 5.

The steps to be carried out by the receiver upon expiration of feedback timer in section 6.2 of [RFC5348] ("action 1") contain the statement:

1) Calculate the average loss event rate using the algorithm described in Section 5.

which is replaced with:

1) Calculate the average loss event rate and average number of lost packets in a loss event using the algorithm described in Section 5.

This statement is added at the beginning of the list of initial steps to take when the first packet is received, in section 6.3 of [RFC5348]:

o Set j = 0.

Section 6.3.1 of [RFC5348] discusses how the loss history is initialized after the first loss event. TFRC approximates the rate to be the maximum value of X_recv so far, assuming that a higher rate introduces loss. Therefore j for this rate is approximated by 1 and the number of packets lost in the first interval is set to 1. This is accomplished by the following change. The first sentence of the fourth paragraph (in section 6.3.1) is:

TFRC does this by finding some value, p, for which the throughput equation in Section 3.1 gives a sending rate within 5% of X_target, given the round-trip time R, and the first loss interval is then set to 1/p.

which is replaced with:

TFRC does this by finding some value, p, for which the throughput equation in Section 3.1 gives a sending rate within 5% of X_target, given the round-trip time R, and j equal to 1. The first loss interval is then set to 1/p.

The second last paragraph in section 6.3.1 ends with:

Thus, the TFRC receiver calculates the loss interval that would be required to produce the target rate X_target of 0.5/R packets per second, for the round-trip time R, and uses this synthetic loss interval for the first loss interval. The TFRC receiver uses 0.5/R packets per second as the minimum value for X_target when initializing the first loss interval.

which is replaced with:

Thus, the TFRC receiver calculates the loss interval that would be required to produce the target rate X_target of 0.5/R packets per second, for the round-trip time R, and for j equal to 1. This synthetic loss interval is used for the first loss interval. The TFRC receiver uses 0.5/R packets per second as the minimum value for X_target when initializing the first loss interval.

2.5. Section 8 of RFC 5348

Section 8.1 explains details about calculating the original TCP throughput equation, which was replaced with a new algorithm in this document. It is therefore obsolete.

2.6. Appendix A of RFC 5348

This section provides a terminology list for TFRC, which is extended as follows:

N: number of emulated TFRC flows.

j: number of packets lost in a loss event.

3. Usage Considerations

The "weighted fairness" service provided by a protocol using MulTFRC is quite different from the service provided by traditional Internet transport protocols. This section intends to answer some questions that this new service may raise.

3.1. Which applications should use MulTFRC?

Like TFRC, MulTFRC is suitable for applications that require a smoother sending rate than standard TCP. Since it is likely that these would be multimedia applications, TFRC has largely been associated with them (and [RFC5348] mentions "streaming media" as an example). Since timely transmission is often more important for them than reliability, multimedia applications usually do not keep retransmitting packets until their successful delivery is ensured. Accordingly, TFRC usage was specified for the Datagram Congestion Control Protocol (DCCP) [RFC4342], but not for reliable data transfers.

MulTFRC, on the other hand, provides an altogether different service. For some applications, a smoother sending rate may not be particularly desirable but might also not be considered harmful, while the ability to emulate the congestion control of N flows may be useful for them. This could include reliable transfers such as the transmission of files. Possible reasons to use MulTFRC for file transfers include the assignment of priorities according to user preferences, increased efficiency with N > 1, the implementation of low-priority "scavenger" services, and resource pooling [Wis+08].

3.2. Setting N

N MUST be set at the beginning of a transfer; it MUST NOT be changed while a transfer is ongoing. The effects of changing N during the lifetime of a MulTFRC session on the dynamics of the mechanism are yet to be investigated; in particular, it is unclear how often N could safely be changed, and how "safely" should be defined in this context. Further research is required to answer these questions.

N is a positive floating point number which can also take values between 0 and 1, making MulTFRC applicable as a mechanism for what has been called a "Lower-than-Best-Effort" (LBE) service. Since it does not reduce its sending rate early as delay increases like some alternative proposals for such a service do (e.g. TCP-LP [Kuz+06], TCP Nice [Ven+02] or 4CP [Liu+07]), it can probably be expected to be more aggressive than these mechanisms if they share a bottleneck at the same time. This also means that MulTFRC is less likely to be prone to starvation. Values between 0 and 1 could also be useful if MulTFRC is used across multiple paths to realize resource pooling [Wis+08].

Setting N to 1 is also possible. In this case, the only difference between TFRC and MulTFRC is that the underlying model of TFRC assumes that all remaining packets following a dropped packet in a "round" (less than one round-trip time apart) are also dropped, whereas the underlying model of MulTFRC does not have this assumption. In other words, other than the equation used in MulTFRC, the underlying equation of TFRC assumes that loss always occurs in bursts. Which choice is better depends on the specific network situation; large windows and other queuing schemes than Drop-Tail seem to make it less likely for the burst-loss assumption to match reality. This document does not make any recommendation about which mechanism to use if only one flow is desired.

Since TCP has been extensively studied, and the aggression of its congestion control mechanism is emulated by TFRC, we can look at the behavior of a TCP aggregate in order to find a reasonable upper limit for N in MulTFRC. From [Alt+06], N TCPs (assuming non-sychronized loss events over connections) can saturate a bottleneck link by roughly 100-100/(1+3N) percent. This means that a single flow can only achieve 75% utilization, whereas 3 flows already achieve 90%. The theoretical gain that can be achieved by adding a flow declines with the total number of flows - e.g., while going from 1 to 2 flows is a 14.3% performance gain, the gain becomes less than 1% beyond 6 flows (which already achieve 95% link utilization). Since the link utilization of MulTFRC can be expected to be roughly the same as the link utilization of multiple TCPs, the approximation above also holds for MulTFRC. Thus, setting N to a much larger value than the values mentioned above will only yield a marginal benefit in isolation but can significantly affect other traffic. Therefore, the maximum value that a user can set for MulTFRC SHOULD NOT exceed 6.

4. Security Considerations

It is well known that a single uncontrolled UDP flow can cause significant harm to a large number of TCP flows that share the same

bottleneck. This potential danger is due to the total lack of congestion control. Because this problem is well known, and because UDP is easy to detect, UDP traffic will often be rate limited by service providers.

If MulTFRC is used within a protocol such as DCCP, which will normally not be considered harmful and will therefore typically not be rate-limited, its tunable aggression could theoretically make it possible to use it for a Denial-of-Service (DoS) attack. In order to avoid such usage, the maximum value of N MUST be restricted. If, as recommended in this document, the maximum value for N is restricted to 6, the impact of MulTFRC on TCP is roughly the same as the impact of 6 TCP flows would be. It is clear that the conjoint congestion control behavior of 6 TCPs is far from being such an attack.

With transport protocols such as TCP, SCTP or DCCP, users can already be more aggressive than others by opening multiple connections. If MulTFRC is used within a transport protocol, this effect becomes more pronounced - e.g., 2 connections with N set to 6 for each of them roughly exhibit the same congestion control behavior as 12 TCP flows. The N limit SHOULD therefore be implemented as a system wide parameter such that the sum of the N values of all MulTFRC connections does not exceed it. Alternatively, the number of connections that can be opened could be restricted.

5. Acknowledgements

This work was partially funded by the EU IST project EC-GIN under the contract STREP FP6-2006-IST-045256.

6. References

6.1. Normative References

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

6.2. Informative References

[All03] Allcock, W., "GridFTP: Protocol Extensions to FTP for the Grid", Open Grid Forum Document GFD.20, 2003.

[Alt+06] Altman, E., Barman, D., Tuffin, B., and M. Vojnovic, "Parallel TCP Sockets: Simple Model, Throughput and Validation", Proceedings of Infocom 2006, April 2006.

[Bri07] Briscoe, B., "Flow rate fairness: dismantling a religion", ACM SIGCOMM Computer Communication Review vol. 37, no. 2, (April 2007), pp. 63-74, April 2007.

[Cro+98] Crowcroft, J. and P. Oechslin, "Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP", ACM SIGCOMM Computer Communication Review vol. 28, no. 3 (July 1998), pp. 53-69, 1998.

[Dam+08] Damjanovic, D., Welzl, M., Telek, M., and W. Heiss, "Extending the TCP Steady-State Throughput Equation for Parallel TCP Flows", University of Innsbruck, Institute of Computer Science, DPS NSG Technical Report 2, August 2008.

[Dam+09] Damjanovic, D. and M. Welzl, "MulTFRC: Providing Weighted Fairness for Multimedia Applications (and others too!)", ACM SIGCOMM Computer Communication Review vol. 39, issue 9 (July 2009), 2009.

[Hac+04] Hacker, T., Noble, B., and B. Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP", Proceedings of Infocom 2004, March 2004.

[Kuo+08] Kuo, F. and X. Fu, "Probe-Aided MulTCP: an aggregate congestion control mechanism", ACM SIGCOMM Computer Communication Review vol. 38, no. 1 (January 2008), pp. 17-28, 2008.

[Kuz+06] Kuzmanovic, A. and E. Knightly, "TCP-LP: low-priority service via end-point congestion control", IEEE/ACM Transactions on Networking (ToN) Volume 14, Issue 4, pp. 739-752., August 2006, <http://www.ece.rice.edu/networks/TCP-LP/>.

[Liu+07] Liu, S., Vojnovic, M., and D. Gunawardena, "Competitive and Considerate Congestion Control for Bulk Data Transfers", Proceedings of IWQoS 2007, June 2007.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.

[Ven+02] Venkataramani, A., Kokku, R., and M. Dahlin, "TCP Nice: a mechanism for background transfers", Proceedings of OSDI '02, 2002.

[Wis+08] Wischik, D., Handley, M., and M. Braun, "The Resource Pooling Principle", ACM Computer Communication Review Volume 38, Issue 5 (October 2008), October 2008, <http://www.cs.ucl.ac.uk/staff/d.wischik/Research/respool.html>.

Authors' Addresses

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no

Dragana Damjanovic
University of Innsbruck
Technikerstr. 21 A
Innsbruck, A-6020
Austria

Phone: +43 512 507 96803
Email: dragana.damjanovic@uibk.ac.at

Bibliography

[1] Internet Capacity Sharing Architecture. http://trac.tools.ietf.org/group/irtf/trac/wiki/CapacitySharingArch.

[2] PlanetLab — An open system for developing, depolying, and accessing planetary-scale services:. http://www.planet-lab.org/.

[3] RED parameters. http://icir.org/floyd/red.html#parameters.

[4] The Network Simulator — ns-2. http://www.isi.edu/nsnam/ns.

[5] M. Ajmone Marsan, G. Carofiglo, M. Garetto, P. Giaccone, E. Leonardi, E. Schiattarella, and A. Tarello. Of Mice and Models. In Quality of Service in Multiservice IP Networks, Third International Workshop, QoS-IP 2005, volume 3375, pages 15–32, Catania, Italy, February 2-4, 2005. Springer. series: Lecture Notes in Computer Science.

[6] M. Ajmone 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 Transactions on Networking, 13(6):1289–1301, 2005.

[7] W. Allcock. GridFTP: Protocol Extensions to FTP for the Grid. Technical report, Open Grid Forum, 2003.

[8] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390 (Proposed Standard), October 2002.

[9] E. Altman, K. Avrachenkov, and C. Barakat. A stochastic model of TCP/IP with stationary random losses. SIGCOMM Computer Communication Review, 30(4):231–242, 2000.

[10] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel TCP Sockets: Simple Model, Throughput and Validation. In Proceedings of IEEE INFOCOM, Barcelona, Spain, April 2006.

[11] E. Altman, R. El Azouzi, D. Ros, and B. Tuffin. Loss Strategies for Competing TCP/IP Connections. In Proceedings of the Networking, pages 926–937. Springer, 2004.

[12] F.M. Anjum and L. Tassiulas. Balanced RED: an algorithm to achieve fairness in the Internet. In Proceedings of IEEE INFOCOM, New York, NY, USA, March 1999.

[13] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: Active Queue Management. IEEE Network, 15(3):48–53, May/Jun 2001. Also in Proceedings of International Teletraffic Congress, Salvador da Bahia, Brazil, September 2001.

[14] F. Baccelli and D. Hong. TCP is Max-Plus Linear and what it tells us on its throughput. SIGCOMM Computer Communication Review, 30(4):219–230, 2000.

[15] F. Baccelli and D. Hong. AIMD, Fairness and Fractal Scaling of TCP Traffic. In Proceedings of IEEE INFOCOM, New York, NY, USA, June 2002.

[16] F. Baccelli and D. Hong. Interaction of TCP Flows as Billiards. IEEE/ACM Transactions on Networking, 13(4):841–853, 2005.

[17] F. Baccelli, D. R. McDonald, and J. Reynier. A Mean-Field Model for Multiple TCP Connections Through a Buffer Implementing RED. Performance Evaluation, 49(1-4):77–97, 2002.

[18] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In ACM SIGCOMM, pages 175–187, Cambridge, Massachusetts, United States, 1999.

[19] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In Proceedings of IEEE INFOCOM, Anchorage, Alaska, USA, April 2001.

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

[21] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.

[22] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In ACM SIGCOMM, pages 24–35, London, UK, August 1994.

[23] B. Briscoe. Flow Rate Fairness: Dismantling a Religion. ACM SIGCOMM Computer Communication Review, 37(2):63–74, April 2007.

[24] B. Briscoe, A. Jacquet, C. Di Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing Congestion Response in an Internetwork Using Re-Feedback. ACM SIGCOMM, Computer Communication Review, 35(4):277–288, August 2005.

[25] E. Brosh, G. Lubetzky-Sharon, and Y. Shavitt. Spatial-Temporal Analysis of passive TCP Measurements. In Proceedings of IEEE INFOCOM, Miami, FL, USA, March 2005.

[26] 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 Proceedings of IEEE INFOCOM, Hong Kong, China, March 2004.

[27] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In Proceedings of IEEE INFOCOM, pages 1742–1751 vol.3, Tel Aviv, Israel, March 2000.

[28] K. Carlberg, P. Gevros, and J. Crowcroft. Lower than Best Effort: a Design and Implementation. In Proceedings ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, pages 244 – 265, San Jose, Costa Rica, 2001.

[29] G. Carofiglio, M. Garetto, E. Leonardi, A. Tarello, and M. Ajmone Marsan. Beyond fluid models: Modelling TCP mice in IP networks under non-stationary random traffic. Computer Networks, 51(1):114–133, 2007.

[30] C. Casetti and M. Meo. A New Approach to Model the Stationary Behavior of TCP Connections. In Proceedings of IEEE INFOCOM, pages 367–375, Tel Aviv, Israel, March 2000.

[31] M. Chen and A. Zakhor. Rate control for streaming video over wireless. In Proceedings of IEEE INFOCOM, Hong Kong, China, March 2004.

[32] M. Chen and A. Zakhor. Flow control over wireless network and application layer implementation. In Proceedings of IEEE INFOCOM, Barcelona, Spain, April 2006.

[33] M. Chen and A. Zakhor. Multiple tfrc connections based rate control for wireless networks. IEEE Transactions on Multimedia, 8(5):1045–1062, 2006.

[34] D.-M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. Computer Networks and ISDN Systems, 17(1):1–14, 1989.

[35] R. Lo Cigno and M. Gerla. Modeling Window Based Congestion Control Protocols with Many Flows. Performance Evaluation, 36-37(1-4):289–306, 1999.

[36] J. Crowcroft and P. Oechslin. Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP. ACM SIGCOMM Computer Communication Review, 28(3):53–69, 1998.

[37] D. Damjanovic, W. Heiss, and M. Welzl. An extension of the TCP steady-state throughput equation for parallel TCP flows, poster. In ACM SIGCOMM, Kyoto, Japan, August 2007.

[38] D. Damjanovic and M. Welzl. Multfrc: providing weighted fairness for multimediaapplications (and others too!). ACM SIGCOMM Computer Communication Review, 39(3):5–12, 2009.

[39] D. Damjanovic, M. Welzl, M. Telek, and W. Heiss. Extending the TCP Steady-State Throughput Equation for Parallel TCP Flows. Technical Report 2, University of Innsbruck, Institute of Computer Science, DPS NSG Technical Report, August 2008.

[40] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In ACM SIGCOMM: Symposium proceedings on Communications architectures & protocols, pages 1–12, Austin, Texas, United States, 1989.

[41] L. Eggert and G. Fairhurst. Unicast UDP Usage Guidelines for Application Designers. RFC 5405 (Best Current Practice), November 2008.

[42] L. Eggert, J. Heidemann, and J. Touch. Effects of Ensemble-TCP. SIGCOMM Computer Communication Review, 30(1):15–29, 2000.

[43] L. Eggert and J. D. Touch. Idletime Scheduling with Preemption Intervals. In A. Herbert and K. P. Birman, editors, 20th ACM Symposium on Operating Systems Principles (SOSP), pages 249–262, Brighton, United Kingdom, October 2005.

[44] V. Firoiu and M. Borden. A Study of Active Queue Management for Congestion Control. In Proceedings of IEEE INFOCOM, pages 1435–1444, Tel Aviv, Israel, March 2000.

[45] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic. ACM Computer Communication Review, 21(5):30–47, 1991.

[46] S. Floyd. Congestion Control Principles. RFC 2914 (Best Current Practice), September 2000.

[47] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[48] S. Floyd and M. Allman. Specifying New Congestion Control Algorithms. RFC 5033 (Best Current Practice), August 2007.

[49] S. Floyd and M. Allman. Comments on the Usefulness of Simple Best-Effort Traffic. RFC 5290 (Informational), July 2008.

[50] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In ACM SIGCOMM, pages 43–56, Stockholm, Sweden, 2000.

[51] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348 (Proposed Standard), September 2008.

[52] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582 (Experimental), April 1999. Obsoleted by RFC 3782.

[53] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking, 1(4):397–413, 1993.

[54] S. Floyd and E. Kohler. TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant. RFC 4828 (Experimental), April 2007.

[55] S. Floyd, E. Kohler, and J. Padhye. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC). RFC 4342 (Proposed Standard), March 2006. Updated by RFC 5348.

[56] M. Garetto, R. Lo Cigno, M. Meo, and M. Ajmone Marsan. Modeling Short-Lived TCP Connections with Open Multiclass Queuing Networks. Computer Networks, 44(2):153–176, 2004.

[57] M. Garetto, R. Lo Cigno, M. Meo, and M. Ajmone Marsan. A Detailed and Accurate Closed Queueing Network Model of Many Interacting TCP Flows. In Proceedings of IEEE INFOCOM, pages 1706–1715, Anchorage, Alaska, USA, April 2001.

[58] T. J. Hacker, B. D. Noble, and B. D. Athey. Improving Throughput and Maintaining Fairness Using Parallel TCP. In Proceedings of IEEE INFOCOM, pages 2480–2489, vol. 4, Hong Kong, China, March 2004.

[59] T. J. Hacker and P. Smith. Stochastic TCP: A Statistical Approach to Congestion Avoidance. In Sixth International Workshop on Protocols for Fast Long Distance Networks (PFLDNeT 2008), Manchester, GB, March 2008.

[60] E. L. Hahne and R. G. Gallager. Round Robin Scheduling for Fair Flow Control in Data Communication Networks. In IEEE International Conference on Communications (ICC), pages 103–107, Toronto, Canada, 1986.

[61] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448 (Proposed Standard), January 2003. Obsoleted by RFC 5348.

[62] M. Hassan and R. Jain. High Performance TCP/IP Networking. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[63] Q. He, C. Dovrolis, and M. Ammar. On the Predictability of Large Transfer TCP Throughput. In ACM SIGCOMM, pages 145–156, New York, NY, USA, 2005.

[64] C. V. Hollot, Y. Liu, V. Misra, and D. F. Towsley. Unresponsive Flows and AQM Performance. In Proceedings of IEEE INFOCOM, San Franciso, CA, USA, March 2003.

[65] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda. Multipath Congestion Control for Shared Bottleneck. In Proceedings of 7th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT), Tokyo, Japan, May 2009.

[66] T. Hutschenreuther and A. Schill. Content Based Discarding in IP-Routers. In International Conference on Computer Communications and Networks (ICCCN), pages 122–126, Las Vegas, USA, Oct 2000.

[67] V. Jacobson. Congestion Avoidance and Control. SIGCOMM Computer Communication Review, 18(4):314–329, August 1988.

[68] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. In Proceedings of IEEE INFOCOM, pages 2490–2501, Hong Kong, China, March 2004.

[69] F. Kelly. Charging and Rate Control for Elastic Traffic. European Transactions on Telecommunications, 8:33–37, 1997.

[70] F. P. Kelly, A.K. Maulloo, and D.K.H. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. Journal of the Operational Research Society, 49:237–252, 1998.

[71] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. ACM SIGCOMM Computer Communication Review, 33(2):83–91, 2002.

[72] P. Key, L. Massoulié, and B. Wang. Emulating Low-Priority Transport at the Application Layer: a Background Transfer Service. SIGMETRICS Performance Evaluation Review, 32(1):118–129, 2004.

[73] W.-J. Kim and B.G. Lee. The FB-RED Algorithm for TCP over ATM. In Proceedings of IEEE GLOBECOM, pages 551–555., Sydney, Australia, November 1998.

[74] E. Kohler, S. Floyd, and A. Sathiaseelan. Faster Restart for TCP Friendly Rate Control (TFRC). Internet draft draft-ietf-dccp-tfrc-faster-restart-06.txt, July 2008.

[75] A. Kumar. Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link. IEEE/ACM Transactions on Networking, 6(4):485–498, 1998.

[76] S. Kunniyur and R. R. Srikant. End-to-End Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks. In Proceedings of IEEE INFOCOM, pages 1323–1332, Tel Aviv, Israel, March 2000.

[77] F.-C. Kuo and X. Fu. Probe-Aided MulTCP: an Aggregate Congestion Control Mechanism. SIGCOMM Computer Communication Review, 38(1):17–28, 2008.

[78] A. Kuzmanovic and E.W. Knightly. TCP-LP: Low-Priority Service via End-Point Congestion Control. IEEE/ACM Transactions on Networking, 14(4):739–752, 2006.

[79] T. V. Lakshman and U. Madhow. Performance Analysis of Window-based Flow Control Using TCP/IP: Effect of High Bandwidth-Delay Products and Random Loss. In Proceedings of the IFIP TC6/WG6.4 Fifth International Conference on High Performance Networking, Grenoble, France, volume C-26 of IFIP Transactions, pages 135–149. North-Holland, 1994.

[80] T. V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. IEEE/ACM Transactions on Networking, 5(3):336–350, 1997.

[81] T. V. Lakshman, Arnold L. Neidhardt, and Teunis J. Ott. The Drop from Front Strategy in TCP and in TCP over ATM. In Proceedings of IEEE INFOCOM, pages 1242–1250, San Francisco, CA, USA, March 1996.

[82] J.-Y. Le Boudec. Rate Adaptation, Congestion Control and Fairness: A Tutorial. http://ica1www.epfl.ch/PS_files/LEB3132.pdf.

[83] C. Lefelhocz, B. Lyles, S. Shenker, and L. Zhang. Congestion Control for Best Effort Service: why we need a new paradigm. IEEE Network, 10(1):10–19, 1996.

[84] D. Lin and R. Morris. Dynamics of Random Early Detection. In ACM SIGCOMM, pages 127–137, Cannes, France, Sept. 1997.

[85] S. Liu, M. Vojnovic, and D. Gunawardena. Competitive and Considerate Congestion Control for Bulk Data Transfers. In Proceedings of Fifteenth IEEE International Workshop on Quality of Service (IWQoS), Evanston, IL, USA, June 2007.

[86] Y. Liu, F. Lo Presti, V. Misra, D. Towsley, and Y. Gu. Fluid Models and Solutions for Large-Scale IP Networks. In SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, volume 31, pages 91–101, New York, NY, USA, June 2003. ACM Press.

[87] S. H. Low. A Duality Model of TCP and Queue Management Algorithms. IEEE/ACM Transactions on Networking, 11(4):525–536, 2003.

[88] S. H. Low and D. E. Lapsley. Optimization Flow Control: Basic Algorithm and Convergence. IEEE/ACM Transactions on Networking, 7(6):861–874, 1999.

[89] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Characterizing and Predicting TCP Throughput on the Wide Area Network. In ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, pages 414–424, Washington, DC, USA, 2005. IEEE Computer Society.

[90] J. K. Mackie-Mason and H. R. Varian. Pricing the Internet. In Public Access to the Internet, pages 269–314. MIT Press, 1995.

[91] J. Mahdavi and S. Floyd. TCP-friendly Unicast Rate-Based Flow Control. Note sent to end2end-interest mailing list, Jan 1997.

[92] A. Mankin, A. Romanow, S. Bradner, and V. Paxson. IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols. RFC 2357 (Informational), June 1998.

[93] M. Mathis. Reflections on the TCP Macroscopic Model. ACM SIGCOMM Computer Communication Review, 39(1):47–49, 2009.

[94] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996.

[95] M. Mathis, J. Semke, and J. Mahdavi. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. ACM Computer Communication Review, 27(3):67 – 82, 1997.

[96] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. ACM SIGCOMM Computer Communication Review, 35(2):37–52, 2005.

[97] M. Mehyar, D. Spanos, and S. H. Low. Optimization Flow Control with Estimation Error. In Proceedings of IEEE INFOCOM, Hong Kong, China, March 2004.

[98] V. Misra, W. Gong, and D. Towsley. Stochastic Differential Equation Modeling and Analysis of TCP Windowsize Behavior. Technical Report ECE-TR-CCS-99-10-01, Department of Electrical and Computer Engineering, University of Massachusetts, November 1999.

[99] 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 ACM SIGCOMM, pages 151–160, Stockholm, Sweden, 2000.

[100] J. Nagle. On packet switches with infinite storage. RFC 970, December 1985.

[101] D. E. Ott, T. Sparks, and K. Mayer-Patel. Aggregate Congestion Control for Distributed Multimedia Applications. In Proceedings of IEEE INFOCOM, Hong Kong, March 2004.

[102] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In Proceedings of IEEE INFOCOM, pages 1346–1355, New York, NY, USA, March 1999.

[103] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a Simple Model and its Empirical Validation. In ACM SIGCOMM, pages 303–314, Vancouver, British Columbia, Canada, 1998.

[104] M. Parris, K. Jeffay, and F. Donelson Smith. Lightweight Active Router-Queue Management for Multimedia Networking. In Multimedia Computing and Networking, SPIE Proceedings Series, pages 162–174, San Jose, CA, USA, January 1999.

[105] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[106] K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481 (Experimental), January 1999. Obsoleted by RFC 3168.

[107] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

[108] E. Raubold and J. D. Haenle. A Method of Deadlock-free Resource Allocation and Flow Control in Packet Networks. In International Conference on Communities and Communications (ICCC), Toronto, Canada, 1976.

[109] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In Proceedings of IEEE INFOCOM, pages 1337–1345, New York, NY, USA, March 21-2 1999.

[110] I. Rhee and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variants. In International Workshop on Protocols for Fast Long Distance Networks (PFLDNeT), Lyon, France, 2005.

[111] I. Rhee and L. Xu. Limitations of Equation-based Congestion Control. In ACM SIGCOMM, pages 49–60, Philadelphia, Pennsylvania, USA, 2005.

[112] R. Rhee, V. Ozdemir, and Y. Yi. TEAR: TCP emulation at receivers – flow control for multimedia streaming. Technical report, Department of Computer Science, NCSU, 2000.

[113] F. Schatz. Reliable data transfer with MulTFRC. Master thesis, Institute of Computer Science, University of Innsbruck, 2009.

[114] S. Sharma and Y. Viniotis. Convergence of a Dynamic Policy for Buffer Management in Shared Buffer ATM Switches. Performance Evaluation, 36-37(1-4):249–266, 1999.

[115] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in Computer Networks: Reshaping the Research Agenda. ACM Computer Communication Review, 26(2):19–43, 1996.

[116] B. Sikdar, S. Kalyanaraman, and K. S. Vastola. An Integrated Model for the Latency and Steady-State Throughput of TCP Connections. Perforance Evaluation, 46(2-3W):139–154, 2000.

[117] M. Singh, P. Pradhan, and P. Francis. MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS. In Proceedings of the 12th IEEE International Conference on Network Protocols, ICNP, pages 129–138, Berlin, Germany, 2004.

[118] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad. Receiver Based Management of Low Bandwidth Access Links. In Proceedings of IEEE INFOCOM, pages 245–254, Tel Aviv, Israel, March 2000.

[119] K. Tan and J. Song. A Compound TCP Approach for High-speed and Long Distance Networks. In Proceedings of IEEE INFOCOM, Barcelona, Spain, April 2006.

[120] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A Mechanism for Background Transfers. ACM SIGOPS Operating Systems Review, OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, 36(SI):329–343, 2002.

[121] M. Vojnovic, J.-Y. Le Boudec, and C. Boutremans. Global Fairness of Additive-Increase and Multiplicative-Decrease with Heterogeneous Round-Trip Times. In Proceedings of IEEE INFOCOM, pages 1303–1312, Tel Aviv, Israel, March 2000. IEEE.

[122] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Transactions on Networking, 14(6):1246–1259, 2006.

[123] M. Welzl. Network Congestion Control: Managing Internet Traffic, Wiley Series on Communications Networking & Distributed Systems. John Wiley & Sons, 2005.

[124] M. Welzl. A Survey of Lower-than-Best Effort Transport Protocols. Internet-draft, http://tools.ietf.org/html/draft-welzl-ledbat-survey-oo.txt, March 2009.

[125] M. Welzl and D. Damjanovic. MulTFRC: TFRC with weighted fairness. Internet-draft draft-welzl-multfrc-01.txt, 2009.

[126] J. Widmer, C. Boutremans, and J.-Y. Le Boudec. End-to-end congestion control for tcp-friendly flows with variable packet size. ACM SIGCOMM Computer Communication Review, 34(2):137–151, 2004.

[127] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. ACM SIGCOMM Computer Communication Review, 31(4):275–285, 2001.

[128] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In Proceedings of IEEE INFOCOM, Hong Kong, China, 2004.

[129] B. Zheng and M. Atiquzzaman. DSRED: an active queue management scheme for next generation networks. In LCN '00: Proceedings of the 25th Annual IEEE Conference on Local Computer Networks, page 242, Tampa, Florida, USA, November 2000. IEEE Computer Society.