Technical University Berlin

Telecommunication Networks Group

Validation of some Ensemble Flow Congestion Management Control Algorithms

Michael Savorić, Holger Karl

{savoric,karl}@ee.tu-berlin.de

Berlin, January 2003

TKN Technical Report TKN-03-003

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract

In a current Internet end system the congestion and flow control of data streams is done individually and separately for all data streams. Some of the data streams of an end system might be connected to the same remote end system or to remote end systems in the same subnetwork. These data streams form a (data stream) set. It is beneficial terms of improving the overall performance, i.e., throughput and fairness, to establish a common congestion control between data streams of a data stream set to continuously share network information between them. These jointly controlled data streams are then combined into a (data stream) ensemble. In [6], we described one such common congestion controller for TCP connections, called ensemble flow congestion management (EFCM) and show its performance benefits.

An open question remained concerning the "aggressiveness" of this controller: Where the performance benefits obtained by simply putting more packets onto the network than a corresponding set of TCP connections would do — in essence, by violating congestion control principle? In this technical report, we consider the algorithms of the current EFCM controller in more detail and show that these algorithms are accurately chosen in terms of aggressiveness to the network. More precisely, we use the current aggregated congestion window size and the current aggregated slow start threshold of an ensemble of n EFCM-controlled TCP connections and compare it with the aggregated congestion window size and the aggregated slow start threshold of a virtual ensemble of n standard TCP connections. We then analytically validate that under certain assumptions EFCM-controlled and standard TCP connections have nearly the same aggregated congestion window size and nearly the same aggregated slow start threshold, i.e., both controllers allow nearly the same overall number of outstanding TCP segments for their TCP connections at any time. Therefore, EFCM-controlled TCP connections are nearly as aggressive to the network as standard TCP connections.

Keywords: TCP, Congestion Control, Flow Control, Network Information Reuse, Common Congestion Control

Contents

Chapter 1

Introduction

1.1 Network information sharing in an Internet end system

Standard TCP implementations perform flow and congestion control separately for each TCP connection, in isolation from all other data streams of an end system. Some of the TCP connections of an end system might be directed to the same remote end system or to remote end systems in the same subnetwork. These TCP connections form a *set*.

As connections of such a set have essentially the same destinations, their packets are likely to use the same data paths within the Internet and hence encounter the same congestion situations during their transmission. It might therefore be advantageous in terms of improving the overall performance of the TCP connections, i.e., the throughput and fairness, to share network information among the TCP connections of belonging to such a set.

In the simplest case, this information sharing happens once: Network information can be shared once between some existing (or recently closed) TCP connections and a new TCP connection. In this case, the available network information is reused to initialize the flow, congestion, and error control variables of the new TCP connection with more adequate values than it is proscribed by standard TCP. However, such one-time information sharing is not the only possibility; a more advanced one is discussed in the next section.

1.2 Common congestion control in an Internet end system

An extension of the one-time network information reuse is the continuous network information sharing among connections belonging to the same set to jointly control them; this approach is called *common congestion control*. TCP connections that actively share network information form an *ensemble* (in a sense, a set describes the potential to become an ensemble). The algorithms that determine which, when, and how network information among the TCP connections of an ensemble is shared form the actual controller of a common congestion control approach.

A common congestion controller's job can be divided into two main tasks: First, a common congestion controller has to manage the one-time network information sharing between *existing* (or *recently closed*) TCP connections of an ensemble and a *new* TCP connection joining this particular ensemble. This task is similar to the controller's job in existing pure network information reuse approaches like the ensemble or temporal TCP control block interdependence (TCBI) approaches [8, 5, 7]. Second, a common congestion controller is responsible for the continuous network information sharing between *concurrent* TCP connections of an ensemble to reach a common congestion control for this ensemble.

These two tasks can be fulfilled in a number of different ways. The main ideas and methods of the four most relevant approaches have been described in [4]. One of these approaches, the ensemble TCP (E-TCP) [2], has been identified as a good basis for our new common congestion control approach called ensemble flow congestion management (EFCM).

1.3 Ensemble flow congestion management (EFCM) controller

In [6], we present the design goals and describe the algorithms of the EFCM controller. The EFCM approach supports standard transport layer interfaces, i.e., sockets. Therefore, EFCM is transparent to all applications and Internet services running on the end system. In previous work, we have shown that the EFCM controller indeed is beneficial for the performance of an ensemble and improves upon the behavior of a set of standard TCP connections. In particular, EFCM not only improves the total throughput of an ensemble, it also ensures that the connections within the ensemble are allocated a fair share of this throughput.

An open question so far is whether these benefits are achieved by behaving more aggressively than standard TCP would do in corresponding situations—which would not be acceptable for a proper coexistence of EFCM and TCP. The intuition behind “aggressiveness” is the amount of traffic that a protocol is allowed to put on the network in a given situation (essentially captured by a connection’s congestion window size and slow start threshold).

1.4 Problem statement: Aggressiveness of EFCM compared to standard TCP

The first question to consider is the level of acceptable aggressiveness of a common congestion controller. The E-TCP [2] approach, for example, requires that an ensemble of n jointly controlled TCP connections must be no more aggressive to the network than a single standard TCP connection. In contrast, the algorithms of the EFCM controller are designed to ensure that an ensemble of n jointly controlled TCP connections will be no more aggressive to the network than *the entire set* of n separately controlled standard TCP connections.

As it turns out, simply stating that EFCM is no more aggressive than TCP (in this sense) is not strictly true. The details are explained in the following chapters, but the intuition is as follows: A set of TCP connections can consist of connections of widely differing aggressiveness. Some connections might have a large current bandwidth allocation (relative size of congestion window), others a small one. If, in such an unbalanced case, an aggressive connection suffers a packet loss, it will reduce its aggressiveness (in particular, the congestion window size). As the total aggressiveness of the set is largely determined by the aggressive connections, the aggregated aggressiveness is vastly reduced. Compared against such a case, an EFCM controller would indeed be more aggressive; it would not reduce the amount of packets to be put on the network as much as TCP would.

This, however, is not a fair comparison to make. As EFCM ensures fairness among TCP connections, its aggressiveness should also be compared to case where the TCP connections fairly share their bandwidth. While standard TCP has no means to ensure such fairness, it is nevertheless a valid, legal, and possible behavior of a set of TCP connections which can happen by pure chance. Hence, any legal behavior of TCP in such a situation is also a legal—for the network acceptable—behavior of a common congestion controller. In fact, we will show that, compared against a fair-share scenario, EFCM is about as aggressive to the network as a set of standard TCP connections.

Moreover, we will show that such a fair sharing is indeed a desirable behavior, even from a throughput point of view. Suppose any distribution of relative throughput share for a set of standard TCP connections is given. After packet losses occur, the total amount of packets to be sent on the network is reduced, depending on which connections are hit by packet losses. How big is the expected value of packets that can be sent to the network after a number of packet losses? We shall show that this expected value is maximized if, initially, all connections have the same relative share of outstanding packets. A large number of packets that a connection set is allowed to send will speed up the transmission of data. Hence, fairness is not only a value in itself but also beneficial for performance.

In summary, our paper uses the following line of argument: First, fairness among standard TCP connections is beneficial for performance. Second, EFCM ensures that an ensemble behaves in a fair manner. Third, an EFCM-controlled ensemble are about as aggressive as a corresponding, *fair* set of TCP-controlled connections.

The remainder of this technical report is organized as follows: Chapter 2 recapitulates and extends the description of the algorithms of the current EFCM controller from [6]. In Chapter 3, we formally define aggressiveness, show that fairness is optimal for standard TCP connections, and demonstrate that EFCM ensembles are about as aggressive as standard TCP connections. Since it is impossible to give an exact analytic model for the dynamic behavior of a set of standard TCP connections, this result is valid only under certain assumptions. Chapter 4 concludes this technical report.

Chapter 2

The EFCM control algorithms

The current version of the EFCM controller [6] performs both one-time network information reuse between existing TCP connections and a new TCP connection similar to the ensemble TCBI approach [7] as well as a common congestion control between concurrently existing TCP connections of an ensemble. In this chapter, we describe the algorithms of the current EFCM controller in detail.

2.1 The TCP variables jointly controlled by EFCM

TCP uses the following variables for congestion control: congestion window size (CWND), slow start threshold (SSTHRESH), round trip time (RTT), smoothed round trip time (SRTT), and round trip time variance (RTTVAR). The first two TCP control variables restrict the load a single TCP connection can send into the network, the last three TCP control variables lead to adequate retransmission timer values for TCP segments sent from a single TCP connection.

The EFCM controller jointly controls the congestion window sizes, the slow start thresholds, the smoothed round trip times, and the round trip time variances of the TCP connections in an ensemble. Hence, the EFCM controller restricts the load the TCP connections of an ensemble can send into the network. In addition, all TCP connections of an ensemble obtain the same value for their retransmission timer.

2.2 The EFCM control algorithms

In the next two sections, we describe the proposed algorithms of the EFCM controller for its two tasks, i.e., initializing the jointly controlled TCP variables of new TCP connections and updating the jointly controlled TCP variables of concurrent TCP connections. To avoid a bursty sending behavior of jointly controlled TCP connections, the EFCM controller uses a rate-based pacing mechanism for consecutive TCP segments. This rate-based pacing mechanism is described in a following section. In addition, the EFCM controller is equipped with a *joint ack clocking* mechanism for every ensemble of TCP connections. This ensemble ack clocking (EAC) allows a fair partitioning of the sent but currently not acknowledged TCP segments, i.e., the outstanding TCP segments, between the TCP connections in an ensemble.

In the following, the values $T_0, T_1, \ldots$ indicate points in time with $T_0 < T_1 < \ldots$ where events occur which have an effect on jointly controlled TCP variables of an ensemble. These ensemble events are: a TCP connection enters or leaves the ensemble, an acknowledgment arrives at a TCP connection of the ensemble, or a packet loss event (retransmission timer timeout, dupack) occurs in one of the TCP connections in the ensemble.

2.2.1 The network information reuse of the EFCM controller for a new TCP connection

If useful network information is available for a new TCP connection, it will reuse this network information and will start with more adequate values for the load the network can cope with and the retransmission timer value. The algorithms of the EFCM controller for the network information reuse between $n-1$ existing TCP connections of an ensemble and a new TCP connection joining this ensemble are described in the following list. Here, T_k is the time when the new connection is established, T_{k-1} is the last point in time where the jointly controlled TCP variables in the ensemble have been adapted due to an ensemble event.

Congestion window size: The EFCM controller computes the sum of all current congestion window sizes of the existing TCP connections of the ensemble plus the standard initial congestion window size of 2, representing the new TCP connection. This sum is used to calculate a fair share, i.e., an arithmetic mean value, of the congestion window size for the TCP connections in the ensemble including the new one. All these TCP connections of the ensemble are assigned this fair share as their new current congestion window size:

$$\begin{aligned} \text{CWND\_AGG}(T_k) &= 2 + \sum_{i=1}^{n-1} \text{CWND}_i(T_{k-1}) \\ \forall i : 1 \le i \le n : \text{CWND}_i(T_k) &= \frac{\text{CWND\_AGG}(T_k)}{n} \end{aligned} \tag{2.1}$$

Slow start threshold: The EFCM controller computes the sum of all current slow start thresholds of the existing TCP connections of the ensemble plus the standard initial slow start threshold 64, representing the new TCP connection. This sum is used to calculate a fair share of the slow start threshold for all TCP connections in the ensemble including the new one. All TCP connections are assigned this slow start threshold fair share as their new current slow start threshold:

$$\begin{aligned} \text{SSTHRESH\_AGG}(T_k) &= 64 + \sum_{i=1}^{n-1} \text{SSTHRESH}_i(T_{k-1}) \\ \forall i : 1 \le i \le n : \text{SSTHRESH}_i(T_k) &= \frac{\text{SSTHRESH\_AGG}(T_k)}{n} \end{aligned} \tag{2.2}$$

Smoothed round trip time: The EFCM controller uses the current value of an aggregated smoothed round trip time of the existing TCP connections of an ensemble as the initial smoothed round trip time of the new TCP connection. If the new TCP connection is the only connection in its ensemble then the initial smoothed round trip time of the new TCP connection is set to the standard value:

$$\text{SRTT}_n(T_k) = \text{SRTT\_AGG}(T_{k-1}) \tag{2.3}$$

Round trip time variance: The EFCM controller uses the current value of an aggregated round trip time variance of the existing TCP connections of an ensemble as the initial round trip time variance of the new TCP connection. If the new TCP connection is the only connection in its ensemble then the initial round trip time variance of the new TCP connection is set to the standard value:

$$\text{RTTVAR}_n(T_k) = \text{RTTVAR\_AGG}(T_{k-1}) \tag{2.4}$$

2.2.2 The common congestion control of the EFCM controller for concurrent TCP connections

Whenever a standard TCP implementation would change the value of one of the four jointly controlled variables of the n TCP connections in an ensemble, EFCM uses this change to trigger updates to this variable to all TCP connections within the same ensemble, according to the following rules:

Congestion window size: After every change of the congestion window size of one of the existing TCP connections in an ensemble, e.g., TCP connection j, an aggregated congestion window size for this ensemble is computed by adding all current congestion window sizes of the TCP connections in the ensemble. The computation for the individual connection follows standard TCP rules. This value is used to calculate a fair share of congestion window size. This congestion window size fair share is the new congestion window size of every TCP connection in an ensemble:

$$\text{CWND}_j(T_k) = \begin{cases} \text{CWND}_j(T_{k-1}) + 1 & \text{if an ACK arrived, SS} \\ \text{CWND}_j(T_{k-1}) + \dfrac{1}{\text{CWND}_j(T_{k-1})} & \text{if an ACK arrived, CA} \\ 1 & \text{a packet loss is detected} \end{cases} \tag{2.5}$$

$$\begin{aligned} \text{CWND\_AGG}(T_k) &= \text{CWND}_j(T_k) + \sum_{1 \le i \le n, i \ne j} \text{CWND}_i(T_{k-1}) \\ \forall i : 1 \le i \le n : \text{CWND}_i(T_k) &= \frac{\text{CWND\_AGG}(T_k)}{n} \end{aligned} \tag{2.6}$$

Slow start threshold: After every change of the slow start threshold of one of the existing TCP connections in an ensemble, e.g., TCP connection i, an aggregated slow start threshold for this ensemble is computed by adding all current slow start thresholds of the TCP connections in the ensemble. This value is used to calculate a fair share of slow start threshold. This value is the new slow start threshold of every TCP connection in an ensemble:

$$\text{SSTHRESH}_j(T_k) = \begin{cases} \text{SSTHRESH}_j(T_{k-1}) & \text{if an ACK arrived} \\ \max\{\text{CWND}_j(T_{k-1})/2, 2\} & \text{a packet loss is detected} \end{cases} \tag{2.7}$$

$$\begin{aligned} \text{SSTHRESH\_AGG}(T_k) &= \text{SSTHRESH}_j(T_k) + \sum_{1 \le i \le n, i \ne j} \text{SSTHRESH}_i(T_{k-1}) \\ \forall i : 1 \le i \le n : \text{SSTHRESH}_i(T_k) &= \frac{\text{SSTHRESH\_AGG}(T_k)}{n} \end{aligned} \tag{2.8}$$

Smoothed round trip time: After every change of the smoothed round trip time of one of the n TCP connections in an ensemble, e.g., TCP connection i, an aggregated smoothed round trip time of this ensemble is updated by a weighted calculation of $(n-1)/n$ times the last value of the aggregated smoothed round trip time plus $1/n$ times the new smoothed round trip time. All TCP connections in an ensemble get this calculation result of the smoothed round trip time as their new smoothed round trip time:

$$\begin{aligned}\text{SRTT\_AGG}(T_k) &= \frac{n-1}{n}\cdot\text{SRTT\_AGG}(T_{k-1})+\frac{1}{n}\cdot\text{SRTT}_j(T_k)\\ \forall i:1\le i\le n:\text{SRTT}_i(T_k) &= \text{SRTT\_AGG}(T_k)\end{aligned}\tag{2.9}$$

Round trip time variance: After every change of the round trip time variance of one of the n TCP connections in an ensemble, e.g., TCP connection i, an aggregated round trip time variance of this ensemble is updated by a weighted calculation of $(n-1)/n$ times the last value of the aggregated round trip time variance plus $1/n$ times the new round trip time variance. All TCP connections in an ensemble get this calculation result of the round trip variance as their new round trip time variance:

$$\begin{aligned}\text{RTTVAR\_AGG}(T_k) &= \frac{n-1}{n}\cdot\text{RTTVAR\_AGG}(T_{k-1})+\frac{1}{n}\cdot\text{RTTVAR}_j(T_k)\\ \forall i:1\le i\le n:\text{RTTVAR}_i(T_k) &= \text{RTTVAR\_AGG}(T_k)\end{aligned}\tag{2.10}$$

If one of the TCP connections in an ensemble is affected by a packet loss, all TCP connections of the ensemble will fairly reduce their congestion window size and slow start threshold to the newly calculated values.

If one of the TCP connections leaves the ensemble, i.e., the TCP connection j has been closed, the current aggregated congestion window size and the current aggregated slow start threshold are fairly shared among the remaining $n-1$ TCP connections in the ensemble.

$$\begin{aligned}\text{CWND\_AGG}(T_k) &= \text{CWND\_AGG}(T_{k-1})\\ \forall i:1\le i\le n, i\ne j:\text{CWND}_i(T_k) &= \frac{\text{CWND\_AGG}(T_k)}{n-1}\end{aligned}\tag{2.11}$$

$$\begin{aligned}\text{SSTHRESH\_AGG}(T_k) &= \text{SSTHRESH\_AGG}(T_{k-1})\\ \forall i:1\le i\le n, i\ne j:\text{SSTHRESH}_u(T_k) &= \frac{\text{SSTHRESH\_AGG}(T_k)}{n-1}\end{aligned}\tag{2.12}$$

2.2.3 The pacing mechanism of the EFCM controller

The pacing mechanism used in the EFCM controller is implemented by using a rate-based mechanism: Every TCP connection in an ensemble can send at most two TCP segments in a burst. The time Δt between two consecutive packet bursts of a TCP connection is calculated by using the aggregated smoothed round trip time and the aggregated congestion window size of an ensemble:

$$\Delta t(T_k)=\alpha_{\text{pacing}}\cdot\text{SRTT\_AGG}(T_k)/\text{CWND\_AGG}(T_k)\tag{2.13}$$

In the current version of the EFCM controller the factor α is set to the fixed value 2 to send at least CWND_AGG TCP segments during one (smoothed) round trip time. Simulation results [6] have shown that this value is too conservative for shorter round trip times, i.e., round trip times much shorter than 100 ms. In these cases, the factor α should be set to smaller values, e.g., 1 or even lower. In fact, the optimal value for the factor α is a function of the current (smoothed) round trip time, i.e.,

$$\alpha = \alpha(\text{SRTT}(T_k)). \tag{2.14}$$

Further research should be done on this topic to find an appropriate mathematical expression for α.

Evidently, the EFCM algorithms impose some overhead in time and space. A rough estimate of the additional time and space complexity of the EFCM controller compared to standard TCP is given in [6].

Chapter 3

Validation of some EFCM control algorithms

3.1 Introduction

In this chapter, we analytically show that the EFCM control algorithms are nearly as aggressive to the network as a set of n standard TCP connections. For this analysis, we use the aggregated congestion window size and the aggregated slow start threshold, i.e, the maximum number of (future) outstanding TCP segments, of the ensemble or the set of TCP connections as a metric for aggressiveness to the network. More specifically, we are interested in the number of packets that either a set of standard TCP connections or a corresponding ensemble can inject into the network. We capture this intuitive notion formally by considering the *expected values* of the aggregated congestion window and slow-start threshold for a set or an ensemble. We will show that the difference between these two values for either the standard or the EFCM-controlled case is small. Formally:

Definition 1 (Virtual ensemble). A set of n TCP connections that *could* form an ensemble if it were under EFCM control is called a *virtual ensemble* of size n.

Definition 2 (Aggregated congestion window and slowstart threshold). For a virtual ensemble of size n, the *virtual aggregated congestion window* and *virtual aggregated slowstart threshold* are the sum of the individual values for each connection.

$$\mathrm{CWND\_AGG}^{(V)} = \sum_{i=1}^{n} \mathrm{CWND}_i \tag{3.1}$$

$$\mathrm{SSTHRESH\_AGG}^{(V)} = \sum_{i=1}^{n} \mathrm{SSTHRESH}_i \tag{3.2}$$

Definition 3 (Aggressiveness). Two flow/congestion control mechanisms are said to have the same aggressiveness if, starting from the same aggregated congestion window and slowstart threshold and experiencing the same sequence of acknowledgment arrivals or packet loss events, their aggregated congestion window and slowstart threshold are still identical.

If they are not identical, the difference in aggregated congestion window and slowstart threshold is a measure of their difference in aggressiveness.

We intend to show that EFCM only marginally differs in aggressiveness from standard TCP.

To simplify our investigation, we do not consider the influence of the pacing and the ensemble ack clocking mechanism of the EFCM controller (cf. Chapter 2) on the aggregated congestion window size or on the aggregated slow start threshold of n jointly controlled TCP connections. We also disregard the algorithms of the EFCM controller that calculate the aggregated smoothed round trip time and the aggregated round trip time variance for these TCP connections.

Aggressiveness after acknowledgment arrival Looking at the description of the EFCM control algorithms from the previous chapter, when an acknowledgment arrives, standard TCP and EFCM perform the same operations on the values of congestion window size and slowstart threshold for a single connection. The aggregated congestion window size of n EFCM-controlled and of n standard TCP connections is increased by at most one, depending on the current aggregated slow start threshold of the ensemble or the slow start threshold of the TCP connection in the set which receives this TCP acknowledgment. This also holds if a number of acknowledgments arrive in a row.

Hence, after an acknowledgment for a so far unacknowledged TCP segment has arrived, the EFCM controller is as aggressive to the network as standard TCP.

Aggressiveness after packet loss event In the case of a single or multiple packet loss event like duplicated acknowledgments or a retransmission timer timeout, the behavior of standard TCP is much more complicated and the comparison with EFCM case is not easy. The remaining sections of this chapter are devoted to answer this question.

In our investigation of standard TCP and EFCM-controlled TCP, we only consider TCP connections which have always TCP segments to send and which have as much outstanding TCP segments as they are allowed to send according to their congestion window size. For simplicity in the following mathematical expressions, we assume that a packet loss event reduces the congestion window size of a TCP connection to the smallest possible value 1, i.e., we do not consider features like fast retransmit or fast recovery [1, 3] of current TCP implementations like TCP Newreno. Otherwise, we have to adapt the following mathematical expressions that we can distinguish between different possible packet loss events and their probabilities. This will be considered in a future version of this technical report.

In the following sections, the values $T_0, T_1, \ldots$ indicate points in time with $T_0 < T_1 < \ldots$. We start our investigation in T_0 and consider the points in time T_l where l consecutive packet loss events occur which have an effect on TCP variables in the set of standard TCP connections or on jointly controlled TCP variables in an ensemble. During this packet loss event burst of length l, no acknowledgments occur in the meantime. These points in time are used as an index to congestion window and slowstart threshold variables, i.e., we will use variables like CWND_AGG$^{(V)}(T_0)$ etc. .

3.2 Standard TCP behavior after consecutive packet losses

For the behavior of a set of standard TCP connections, we have to establish two results: one are expressions of the expected values for (aggregated) congestion windows and slow start threshold, the second is a consideration about how the relative behavior of these connections impacts the expected value of congestion window / slow start threshold after a number of consecutive packet loss events occur.

3.2.1 Expected values for aggregated congestion windows and slow start thresholds

We start our investigation of a set of n standard TCP connections at time T_0. After a packet loss event, e.g., a retransmission timer timeout, at time T_1 in TCP connection j of the set, the basic algorithms for the congestion window size and slow start threshold computation in standard TCP work as follows [1]:

$$\forall i : 1 \leq i \leq n : \mathrm{CWND}_i(T_1) = \begin{cases} 1 & i = j \\ \mathrm{CWND}_i(T_0) & i \neq j \end{cases} \tag{3.3}$$

$$\forall i : 1 \leq i \leq n : \mathrm{SSTHRESH}_i(T_1) = \begin{cases} \max\{\mathrm{CWND}_i(T_0)/2, 2\} & i = j \\ \mathrm{SSTHRESH}_i(T_0) & i \neq j \end{cases} \tag{3.4}$$

In the following, we investigate how the aggregated congestion window size and the aggregated slow start threshold of a set of n standard TCP connections decreases after a burst of l consecutive packet loss events occurs. For the analysis of standard TCP, we use a combinatorial model that describes how the l consecutive packet loss events are distributed over the n standard TCP connections of the set. For this combinatorial model, we assume that

- all congestion window sizes $\mathrm{CWND}_i(T_0)$ and slow start thresholds $\mathrm{SSTHRESH}_i(T_0)$ of the set of n TCP connections are integers,
- the probability p for each outstanding TCP segment to get lost is equal to all other outstanding TCP segments of the set of n standard TCP connections,
- all time-dependencies regarding the point in time when each of the outstanding TCP segments is sent and when a packet loss event that we can assign to this TCP segment, e.g., a retransmission timer timeout, will occur are neglected.

Under these assumptions, we can use an urn model that gives us probabilities for the distribution of l consecutive packet loss events among n TCP connections in a set: If $\mathrm{CWND\_AGG}^{(V)}(T_0)$ TCP segments are outstanding, the probability $p_{i,k,l}$, $0 \leq i \leq n$, $0 \leq k \leq l \leq l_{\max} = \mathrm{CWND\_AGG}^{(V)}(T_0)$, that TCP connection i is affected by k of the l consecutive packet loss events, follows the hypergeometrical distribution:

$$p_{i,k,l} = \frac{\binom{\mathrm{CWND}_i(T_0)}{k} \cdot \binom{\mathrm{CWND\_AGG}^{(V)}(T_0) - \mathrm{CWND}_i(T_0)}{l-k}}{\binom{\mathrm{CWND\_AGG}^{(V)}(T_0)}{l}}. \tag{3.5}$$

The analogy for this formula is the following problem: suppose there are $\mathrm{CWND}_i(T_0)$ red balls and $\mathrm{CWND\_AGG}^{(V)}(T_0) - \mathrm{CWND}_i(T_0)$ blue balls in an urn. What is the probability that red appears k times if balls are drawn l times without replacement?

The probability that each TCP connection i of the n TCP connections in the set is affected by exactly k_i of the l consecutive packet loss events, $k_1 + \cdots + k_n = l$, is a generalized hypergeometrical distribution:

$$p(l, k_1, \ldots, k_n) = \frac{\binom{\text{CWND}_1(T_0)}{k_1} \cdot \ldots \cdot \binom{\text{CWND}_n(T_0)}{k_n}}{\binom{\text{CWND\_AGG}^{(V)}(T_0)}{l}}. \tag{3.6}$$

The analogy for this formula is the following problem: suppose there are $\text{CWND\_AGG}^{(V)}(T_0)$ balls of n different colors in an urn, for color i we have $\text{CWND}_i(T_0)$ balls in the urn. What is the probability that a specific color i appears k times if balls are drawn l times without replacement?

Starting from these formulas, we want to deduce expected values for the aggregated congestion window size and the aggregated slow start threshold after every of the l consecutive packet loss events.

After l consecutive packet loss events, the expected value of the congestion window size $\text{CWND}_i(T_l)$ of TCP connection i can be computed as follows:

$$\text{E}\left[\text{CWND}_i(T_l)\right] = p_{i,0,l} \cdot \text{CWND}_i(T_0) + \sum_{j=1}^{\text{CWND}_i(T_0)} p_{i,j,l} \cdot 1 = p_{i,0,l} \cdot \text{CWND}_i(T_0) + (1 - p_{i,0,l}). \tag{3.7}$$

After l consecutive packet loss events, the expected value of the aggregated congestion window size $\text{CWND\_AGG}^{(V)}(T_l)$ of the set of n TCP connections can be computed as follows:

$$\begin{aligned} \text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right] &= \sum_{k_1 + \ldots + k_n = l} p(l, k_1, \ldots, k_n) \cdot \left(\sum_{1 \le i \le n : k_i > 0} 1 + \sum_{1 \le i \le n : k_i = 0} \text{CWND}_i(T_0) \right) \\ &= \sum_{i=1}^{n} \text{E}\left[\text{CWND}_i(T_l)\right]. \end{aligned} \tag{3.8}$$

Equation (3.8) shows that the expected value of the aggregated congestion window size after l consecutive packet loss events can be calculated in two different ways: by a sum over all possibilities of valid packet loss distributions over the n TCP connections in the set or by a sum over the expected congestion window sizes of each of the n TCP connections in the set.

With the same considerations, we can deduce formulas for the expected values of the slow start threshold of a single TCP connection and the aggregated slow start threshold of the set of n TCP connections:

$$\begin{aligned} \text{E}\left[\text{SSTHRESH}_i(T_l)\right] &= p_{i,0,l} \cdot \text{SSTHRESH}_i(T_0) + \\ &\quad p_{i,1,l} \cdot \max\left\{\text{CWND}_i(T_0)/2, 2\right\} + 2 \cdot (1 - p_{i,0,l} - p_{i,1,l}) \end{aligned} \tag{3.9}$$

and

$$\begin{aligned}
\mathrm{E}\left[\mathrm{SSTHRESH\_AGG}^{(V)}(T_l)\right] &= \sum_{k_1+\ldots+k_n=l} p(l,k_1,\ldots,k_n)\cdot \\
&\left(\sum_{1\le i\le n:k_i>1} 2 + \sum_{1\le i\le n:k_i=1} \max\{\mathrm{CWND}_i(T_0)/2, 2\} + \sum_{1\le i\le n:k_i=0} \mathrm{SSTHRESH}_i(T_0)\right) \\
&= \sum_{i=1}^{n} \mathrm{E}\left[\mathrm{SSTHRESH}_i(T_l)\right]. \qquad (3.10)
\end{aligned}$$

3.2.2 Relative behavior of TCP connections

These values for the expected aggregated congestion window size and slow start threshold serve as a starting point for a consideration about the relative fairness of n TCP connections.

For a set of n standard TCP connections, there is no mechanism that would ensure fairness between them: one connection could have a large amount of outstanding packets, another one only a small number. In essence, the aggregated congestion window size and slow start threshold can be arbitrarily shared between this set of TCP connections. Essentially, any combination of congestion window sizes $\mathrm{CWND}_i(T_0)$ and slow start thresholds $\mathrm{SSTHRESH}_i(T_0)$ that meet the following equations

$$\forall i : 1 \le i \le n : 1 \le \mathrm{CWND}_i(T_0) \le \mathrm{CWND\_AGG}^{(V)}(T_0) - (n-1) \wedge \qquad (3.11)$$

$$\mathrm{CWND\_AGG}^{(V)}(T_0) = \sum_{i=1}^{n} \mathrm{CWND}_i(T_0) \qquad (3.12)$$

$$\forall i : 1 \le i \le n : 2 \le \mathrm{SSTHRESH}_i(T_0) \le \mathrm{SSTHRESH\_AGG}^{(V)}(T_0) - 2\cdot(n-1) \wedge$$

$$\mathrm{SSTHRESH\_AGG}^{(V)}(T_0) = \sum_{i=1}^{n} \mathrm{SSTHRESH}_i(T_0) \qquad (3.13)$$

represents a legal instance of TCP behavior. But how do these values influence the behavior of a set of connections after a burst of l consecutive packet loss events occurred? More specifically: Starting from any fixed set of $\mathrm{CWND}_i(T_0)$'s and $\mathrm{SSTHRESH}_i(T_0)$'s, how does the expected value for aggregated congestion window sizes and slow start threshold look like after l consecutive packet loss events? We intend to show that the expected aggregated congestion window size after errors is maximized in nearly all cases if $|\mathrm{CWND}_i(T_0) - \mathrm{CWND}_j(T_0)| \le 1$—the result does not hold in general as there are some extreme combinations of values for which this property is slightly violated; numbers are given later.

In addition, we consider the *overall* expected aggregated congestion window size:

$$\begin{aligned}
\mathrm{E}\left[\mathrm{CWND\_AGG}^{(V)}\right] &= \sum_{l=0}^{l_{\max}} \Pr[X=l]\cdot \mathrm{E}\left[\mathrm{CWND\_AGG}^{(V)}(T_l)\right] \qquad (3.14) \\
&= \sum_{l=0}^{l_{\max}} \mathrm{gmp}(l+1,p)\cdot \mathrm{E}\left[\mathrm{CWND\_AGG}^{(V)}(T_l)\right], \qquad (3.15)
\end{aligned}$$

where p is the packet loss probability p in the Internet and X is the random variable representing the number of consecutive packet losses — X is geometrically distributed with parameter p, $\Pr[X = l] = \text{gmp}(l+1, p)$ — and $l_{\max} = \text{CWND\_AGG}^{(V)}(T_0)$ is the maximum number of outstanding and hence potentially lost packets. We will show that this overall expected value is maximized if $|\text{CWND}_i(T_0) - \text{CWND}_j(T_0)| \leq 1, \forall i, j : 1 \leq i, j \leq n$.

We will also show that the expected aggregated slow start threshold after a burst of l packet loss events is independent from the partition of the current aggregated slow start threshold $\text{SSTHRESH\_AGG}^{(V)}(T_0)$ if $\text{CWND}_i(T_0) = \text{CWND}_j(T_0), \forall i, j : 1 \leq i, j \leq n$.

Lemma 1. *The expected aggregated congestion window size after l packet loss events $E[CWND\_AGG^{(V)}(T_l)]$ is maximized in nearly all cases if $|CWND_i(T_0) - CWND_j(T_0)| \leq 1$. The expected overall aggregated congestion window size $E[CWND\_AGG^{(V)}]$ is maximized if $|CWND_i(T_0) - CWND_j(T_0)| \leq 1, \forall i, j : 1 \leq i, j \leq n$.*

Proof. We have n TCP connections in our set, each of them with a current congestion window size $\text{CWND}_i(T_0)$, $1 \leq i \leq n$, and the current aggregated congestion window size $\text{CWND\_AGG}^{(V)}(T_0) = \text{CWND}_1(T_0) + \ldots + \text{CWND}_n(T_0)$ of the set. What we investigate is how the expected aggregated congestion window size of the set of n TCP connections is influenced by l consecutive packet loss events, $1 \leq l \leq l_{\max} = \text{CWND\_AGG}^{(V)}(T_0)$, depending on the choice of the current congestion window sizes, i.e., we consider the following mathematical expression:

$$\text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right] = f(l, \text{CWND}_1(T_0), \ldots, \text{CWND}_n(T_0)) \tag{3.16}$$

To reach this aim, we consider only two TCP connections of the whole set, w.l.o.g., TCP connection 1 and TCP connection 2, in more detail. These two TCP connections have an aggregated congestion window size

$$\text{CWND\_AGG}^{*(V)}(T_0) = \text{CWND}_1(T_0) + \text{CWND}_2(T_0) \tag{3.17}$$

and can be affected by l consecutive packet loss events, with $1 \leq l \leq l^*_{\max} = \text{CWND\_AGG}^{*(V)}(T_0)$. We investigate how their expected aggregated congestion window size

$$\text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right] = f(l, \text{CWND}_1(T_0), \text{CWND}_2(T_0)) \tag{3.18}$$

after l consecutive packet loss events depends on the choice of the current congestion window sizes $\text{CWND}_1(T_0)$ and $\text{CWND}_2(T_0)$.

The probability $p_{k,l}$ that k of the l packet loss events occur in TCP connection 1 and $l-k$ of the l packet loss events occur in TCP connection 2, is:

$$p_{k,l} = \frac{\binom{\text{CWND}_1(T_0)}{k} \cdot \binom{\text{CWND}_2(T_0)}{l-k}}{\binom{\text{CWND}_1(T_0)+\text{CWND}_2(T_0)}{l}} \tag{3.19}$$

Let T_l be the point in time where the l-th consecutive packet loss event occurs. The expected aggregated congestion window size $\text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right]$ after l packet loss events, is:

$$\begin{aligned}\text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right] &= p_{l,l} \cdot (1 + \text{CWND}_2(T_0)) + \\ &\quad p_{0,l} \cdot (1 + \text{CWND}_1(T_0)) + 2 \cdot (1 - p_{l,l} - p_{0,l})\end{aligned} \tag{3.20}$$

In the following, we fix the current aggregated congestion window size CWND_AGG$^{*(V)}(T_0)$ and investigate how the expected aggregated congestion window size E[CWND_AGG$^{*(V)}(T_l)$] changes under different partitions of CWND_AGG$^{*(V)}(T_0)$, i.e., we consider all possible cases

$$\begin{aligned} 1 \le \text{CWND}_1(T_0), \text{CWND}_2(T_0) \le \text{CWND\_AGG}^{*(V)}(T_0) - 1 : \\ \text{CWND}_1(T_0) + \text{CWND}_2(T_0) = \text{CWND\_AGG}^{*(V)}(T_0) \end{aligned} \tag{3.21}$$

Numerical calculations (see Appendix A) show that in nearly all cases the expected aggregated congestion window size E[CWND_AGG$^{*(V)}(T_l)$] after l consecutive packet loss events reaches its maximum, if CWND$_1(T_0)$ = CWND$_2(T_0)$ for even CWND_AGG$^{*(V)}(T_0)$ or $|\text{CWND}_1(T_0) - \text{CWND}_2(T_0)| = 1$ for odd CWND_AGG$^{*(V)}(T_0)$. There exist only a few other partitions of the current aggregated congestion window size which have a slightly larger expected aggregated congestion window size after a specific number of consecutive packet loss events. For example, if we have a current aggregated congestion window size of 32 and consider two partitions $P_1 = (\text{CWND}_1(T_0) = 16, \text{CWND}_2(T_0) = 16)$ and $P_2 = (2, 30)$, the expected aggregated congestion window size after 17 consecutive packet loss events is exactly 2 for partition P_1 and ≈ 2.2117 as the maximum possible value for partition P_2.

But

$$\text{E}\left[\text{CWND\_AGG}^{*(V)}\right] = \sum_{l=0}^{l^*_{\max}} \text{gmp}(l+1, p) \cdot \text{E}\left[\text{CWND\_AGG}^{*(V)}(T_l)\right], \tag{3.22}$$

with $l^*_{\max} = \text{CWND\_AGG}^{*(V)}(T_0)$, is maximized only if CWND$_1(T_0)$ = CWND$_2(T_0)$ for even CWND_AGG$^{*(V)}(T_0)$ or if $|\text{CWND}_1(T_0) - \text{CWND}_2(T_0)| = 1$ for odd CWND_AGG$^{*(V)}(T_0)$ For the example above and an exemplarily chosen packet loss probability of $p = 10^{-4}$ in the Internet, we get ≈ 31.9985 for partition P_1 as the maximum possible value and ≈ 31.9973 for partition P_2 as the third smallest possible value. Of course, in this consideration, the expected value is dominated by the value for the error-free case as the packet loss probability is here rather small.

Therefore, to reach the maximum expected overall aggregated congestion window size the current aggregated congestion window size of the two TCP connections has to be fairly partitioned, e.g., CWND$_1(T_0)$ = CWND$_2(T_0)$ for even CWND_AGG$^{*(V)}(T_0)$. Since we have chosen the two TCP connections w.l.o.g., this result is valid for any pair of two TCP connections of the set. Hence, the expected overall aggregated congestion window size E[CWND_AGG$^{(V)}$] reaches its maximum only for CWND$_1(T_0) = \ldots =$ CWND$_n(T_0)$ if CWND_AGG$^{(V)}(T_0)$ is divisible by n without remainder or for $\forall i, j : 1 \le i, j \le n : |\text{CWND}_i(T_0) - \text{CWND}_j(T_0)| \le 1$ in all other cases. $\square$

Lemma 2. *In the special case CWND$_i(T_0)$ = CWND$_j(T_0)$ = CWND(T_0), $\forall i, j : 1 \le i, j \le n$, the expected value of the aggregated slow start threshold after l consecutive packet loss events is independent of the partition of the current aggregated slow start threshold SSTHRESH_AGG$^{(V)}(T_0)$.*

Proof. In the case of a fairly shared current aggregated congestion window size, the probability that TCP connection i is affected by k of the l packet loss events is equal to the probability that TCP connection j is affected by k of the l packet loss events, i.e. (cf. Equation 3.5):

$$\forall i, j : 1 \le i, j \le n : p_{i,k,l} = p_{j,k,l} = p_{k,l} \tag{3.23}$$

It follows (cf. Equations 3.9 and 3.10):

$$\begin{aligned}
&\mathrm{E}\left[\text{SSTHRESH\_AGG}^{(V)}(T_l)\right] = \sum_{i=1}^{n} \mathrm{E}\left[\text{SSTHRESH}_i(T_l)\right] = \\
&\sum_{i=1}^{n} \left(p_{0,l} \cdot \text{SSTHRESH}_i(T_0) + p_{1,l} \cdot \max\left\{\text{CWND}(T_0)/2, 2\right\} + 2 \cdot (1 - p_{0,l} - p_{1,l})\right) = \\
&p_{0,l} \cdot \sum_{i=1}^{n} \text{SSTHRESH}_i(T_0) + p_{1,l} \cdot n \cdot \max\left\{\text{CWND}(T_0)/2, 2\right\} + 2 \cdot n \cdot (1 - p_{0,l} - p_{1,l}) = \\
&p_{0,l} \cdot \text{SSTHRESH\_AGG}^{(V)}(T_0) + p_{1,l} \cdot \max\left\{\text{CWND\_AGG}^{(V)}(T_0)/2, 2 \cdot n\right\} + \\
&2 \cdot n \cdot (1 - p_{0,l} - p_{1,l}) \qquad (3.24)
\end{aligned}$$

Since Equation 3.24 is independent of the partition of the current aggregated slow start threshold $\text{SSTHRESH\_AGG}^{(V)}(T_0)$, the lemma holds. □

From the end system's point of view, the special case, i.e. the case where all congestion window sizes and slow start thresholds of a set of TCP connections are equal, is the best case for a set of n standard TCP connections regarding the expected overall aggregated congestion window size. The special case is also the expected case of a set of n standard TCP connections regarding the aggregated slow start threshold. From the network's point of view, the special case represents one possible standard-conforming congestion control behavior of n separately controlled TCP connections of an end system.

3.3 EFCM-controlled TCP behavior after consecutive packet losses

We start our investigation of an ensemble of n EFCM-controlled TCP connections at time T_0. The variables $\text{CWND\_AGG}(T_0)$ and $\text{SSTHRESH\_AGG}(T_0)$ are the current aggregated values for the congestion window sizes and the slow start thresholds of n EFCM-controlled TCP connections in an ensemble. The variables $\text{CWND}_i(T_0)$ and $\text{SSTHRESH}_i(T_0)$ are the current values for the congestion window size and the slow start threshold of a single TCP connection in that ensemble:

$$\text{CWND\_AGG}(T_0) = \sum_{i=1}^{n} \text{CWND}_i(T_0) \qquad (3.25)$$

$$\text{SSTHRESH\_AGG}(T_0) = \sum_{i=1}^{n} \text{SSTHRESH}_i(T_0) \qquad (3.26)$$

and $\forall i, j : 1 \leq i, j \leq n$:

$$\text{CWND}_i(T_0) = \text{CWND}_j(T_0) \qquad (3.27)$$

$$\text{SSTHRESH}_i(T_0) = \text{SSTHRESH}_j(T_0) \qquad (3.28)$$

After a packet loss event, e.g., a retransmission timer timeout, in TCP connection j of the ensemble, the basic algorithms for the congestion window size and slow start threshold computation in the EFCM controller work as follows (cf. Section 2.2.2):

$$\begin{aligned} \text{CWND\_AGG}(T_1) &= \text{CWND\_AGG}(T_0) - \text{CWND}_j(T_0) + 1 \\ \forall i : 1 \leq i \leq n : \text{CWND}_i(T_1) &= \frac{\text{CWND\_AGG}(T_1)}{n} \qquad (3.29) \\ & \qquad (3.30) \end{aligned}$$

or writing this as a recursive formula:

$$\begin{aligned} \text{CWND\_AGG}(T_{l+1}) &= \text{CWND\_AGG}(T_l) - \frac{\text{CWND\_AGG}(T_l)}{n} + 1 \qquad (3.31) \\ & \qquad (3.32) \end{aligned}$$

and solving for l yields

$$\begin{aligned} \text{CWND\_AGG}(T_l) &= \left(1 - \frac{1}{n}\right)^l \cdot \text{CWND\_AGG}(T_0) + \sum_{i=0}^{l-1} \left(1 - \frac{1}{n}\right)^i \qquad (3.33) \\ &= \left(1 - \frac{1}{n}\right)^l \cdot \text{CWND\_AGG}(T_0) + n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^l\right) \qquad (3.34) \end{aligned}$$

The slowstart threshold behaves as follows:

$$\begin{aligned} \text{SSTHRESH\_AGG}(T_1) &= \text{SSTHRESH\_AGG}(T_0) - \text{SSTHRESH}_j(T_0) + \\ & \quad \max\{\text{CWND}_j(T_0)/2, 2\} \qquad (3.35) \\ \forall i : 1 \leq i \leq n : \text{SSTHRESH}_i(T_1) &= \frac{\text{SSTHRESH\_AGG}(T_1)}{n} \qquad (3.36) \end{aligned}$$

Note that these are *not* expected values, but the precise values—there is no random element as it does not matter which connection encounters the packet loss. An additional advantage of EFCM is thus is much simpler and easier to analyze behavior.

If l consecutive packet loss events occur in an ensemble of n TCP connections, the values CWND_AGG(T_l) and SSTHRESH_AGG(T_l) for the aggregated congestion window size and the aggregated slow start threshold after the packet loss event burst have to be numerically determined by an l-fold iteration of this algorithm. This has been done with a tcl-script (cf. Appendix A).

3.4 Comparison of standard TCP and EFCM-controlled TCP

In this section, we compare the analytical results from the special case of n standard TCP connections in a set with the numerical results of n EFCM-controlled TCP connections in an ensemble. In the following Tables 3.1 and 3.2, we exemplarily show the analytical and numerical results for a current congestion window size of 32 and a current slow start threshold of 64 for each of the n TCP connections in the set or in the ensemble before the packet loss event burst occur, i.e.:

$$\begin{aligned} \text{CWND\_AGG}^{(V)}(T_0) = \text{CWND\_AGG}(T_0) &= n \cdot 32 \\ \text{SSTHRESH\_AGG}^{(V)}(T_0) = \text{SSTHRESH\_AGG}(T_0) &= n \cdot 64 \end{aligned}$$

3.4.1 Aggregated congestion window size

Using the tcl-scripts from Appendix A, we have compared the aggregated congestion window size of standard TCP and of EFCM-controlled TCP connections. In this comparison, we have varied the number n of TCP connections and the current congestion window sizes of the TCP connections before the packet loss event burst occur. We achieve the following result: For every number l of consecutive packet loss events the expected aggregated congestion window size of n standard TCP connections in a set is equal to or slightly lower than the aggregated congestion window size of n EFCM-controlled TCP connections in an ensemble, i.e.:

$$\forall l : \text{E}\left[\text{CWND\_AGG}^{(V)}(T_l)\right] \leq \text{CWND\_AGG}(T_l) \tag{3.37}$$

The absolute differences $\Delta_{n,l}^{\text{CWND}}$ of the aggregated congestion window sizes of a set of n standard TCP connections and an ensemble of n EFCM-controlled TCP connections after l consecutive packet loss events are shown for the example case in the following Table 3.1. For other current congestion window sizes and for other current slow start thresholds the results are comparable to the stated ones. For a single EFCM-controlled TCP connection and for realistic numbers of jointly controlled TCP connections the congestion window size is at most 0.30 larger than the congestion window size of a single standard TCP connection.

3.4.2 Aggregated slow start threshold

Using the tcl-scripts from Appendix A, we have compared the aggregated slow start threshold of standard TCP and of EFCM-controlled TCP connections. In this comparison, we have varied the number n of TCP connections, the current congestion window sizes, and the current slow start thresholds of the TCP connections before the packet loss event burst occur. The absolute differences $\Delta_{n,l}^{\text{SSTHRESH}}$ of the aggregated congestion window sizes of a set of n standard TCP connections and n EFCM-controlled TCP connections after l consecutive packet loss events are shown for the example case in the following Table 3.2.

We have performed extensive investigations with various realistic current congestion window sizes in the range from 1 to 128, with various realistic current slow start thresholds in the range from 2 to 64, and with different numbers n of standard TCP connections in a set and EFCM-controlled TCP connections in an ensemble. For most other current congestion window sizes and current slow start thresholds the results are comparable to the stated ones. Only in some academic cases, where the current congestion window sizes are very low and the current slow start thresholds are very high, e.g., all n TCP connections of the set or ensemble are recently started, the aggregated slow start threshold of n EFCM-controlled TCP connections is (much) more reduced than the aggregated slow start threshold of n standard TCP connections. But since the congestion window sizes of n EFCM-controlled TCP connections are decreased comparably as the congestion window sizes of n standard

Table 3.1: Analytical and numerical results for the aggregated congestion window size of a set of n standard TCP connections and an ensemble of n EFCM-controlled TCP connections — after l consecutive packet loss events $\Delta^{\text{CWND}}_{n,l}$ denotes the difference of the expected aggregated congestion window size of a set of n standard TCP connections compared to the aggregated congestion window size of an ensemble of n EFCM-controlled TCP connections

l	$\Delta^{\text{CWND}}_{1,l}$	$\Delta^{\text{CWND}}_{2,l}$	$\Delta^{\text{CWND}}_{3,l}$	$\Delta^{\text{CWND}}_{4,l}$	$\Delta^{\text{CWND}}_{5,l}$	$\Delta^{\text{CWND}}_{10,l}$	$\Delta^{\text{CWND}}_{20,l}$	$\Delta^{\text{CWND}}_{50,l}$
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	−0.24603	−0.21754	−0.18307	−0.15597	−0.08746	−0.04609	−0.01900
3	0.00000	−0.36905	−0.43663	−0.41336	−0.37552	−0.23658	−0.13148	−0.05588
4	0.00000	−0.36602	−0.58269	−0.62142	−0.60227	−0.42657	−0.25004	−0.10957
5	0.00000	−0.29998	−0.64625	−0.77746	−0.80431	−0.64083	−0.39627	−0.17904
6	0.00000	−0.21938	−0.64330	−0.87423	−0.96594	−0.86629	−0.56518	−0.26329
7	0.00000	−0.14845	−0.59601	−0.91627	−1.08183	−1.09281	−0.75231	−0.36137
8	0.00000	−0.09485	−0.52441	−0.91334	−1.15301	−1.31268	−0.95368	−0.47237
9	0.00000	−0.05795	−0.44368	−0.87668	−1.18401	−1.52021	−1.16572	−0.59540
10	0.00000	−0.03414	−0.36391	−0.81698	−1.18108	−1.71135	−1.38527	−0.72963
15	0.00000	−0.00167	−0.09897	−0.42975	−0.88078	−2.35926	−2.50840	−1.54157
20	0.00000	−0.00006	−0.01953	−0.16920	−0.49864	−2.51091	−3.51882	−2.52548
25	0.00000	0.00000	−0.00319	−0.05618	−0.24114	−2.32104	−4.30274	−3.60958
30	0.00000	0.00000	−0.00046	−0.01662	−0.10491	−1.96110	−4.82650	−4.73699
35	0.00000	0.00000	−0.00006	−0.00452	−0.04224	−1.55569	−5.10169	−5.86325
40	0.00000	0.00000	−0.00001	−0.00116	−0.01603	−1.17715	−5.16267	−6.95432
45	0.00000	0.00000	0.00000	−0.00029	−0.00581	−0.85831	−5.05270	−7.98480
50	0.00000	0.00000	0.00000	−0.00007	−0.00203	−0.60726	−4.81564	−8.93642
60	0.00000	0.00000	0.00000	0.00000	−0.00023	−0.28293	−4.11436	−10.55809
70	0.00000	0.00000	0.00000	0.00000	−0.00002	−0.12234	−3.30455	−11.77141
80	0.00000	0.00000	0.00000	0.00000	0.00000	−0.04994	−2.53393	−12.57794
90	0.00000	0.00000	0.00000	0.00000	0.00000	−0.01949	−1.87357	−13.00907
100	0.00000	0.00000	0.00000	0.00000	0.00000	−0.00734	−1.34491	−13.11227

TCP connections even in these cases, the overall possible number of outstanding TCP segments after the packet loss event burst is similar until new TCP acknowledgments arrive. Then, the running EFCM-controlled TCP connections will increase their aggregated congestion window size (much) faster than the running standard TCP connections, but much lower than new standard TCP connections will do. Therefore, the EFCM controller does not violate any congestion control mechanisms of standard TCP.

Table 3.2: Analytical and numerical results for the aggregated slow start threshold of a set of n standard TCP connections and an ensemble of n EFCM-controlled TCP connections — after l consecutive packet loss events $\Delta^{\text{SSTHRESH}}_{n,l}$ denotes the difference of the expected aggregated slow start threshold of a set of n standard TCP connections compared to the aggregated slow start threshold of an ensemble of n EFCM-controlled TCP connections

l	$\Delta^{\text{SSTHRESH}}_{1,l}$	$\Delta^{\text{SSTHRESH}}_{2,l}$	$\Delta^{\text{SSTHRESH}}_{3,l}$	$\Delta^{\text{SSTHRESH}}_{4,l}$	$\Delta^{\text{SSTHRESH}}_{5,l}$	$\Delta^{\text{SSTHRESH}}_{10,l}$	$\Delta^{\text{SSTHRESH}}_{20,l}$	$\Delta^{\text{SSTHRESH}}_{50,l}$
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	+ 0.48016	+ 0.26140	+ 0.17421	+ 0.12893	+ 0.05408	+ 0.02445	+ 0.00916
3	0.00000	+ 0.92857	+ 0.59023	+ 0.42278	+ 0.32641	+ 0.14887	+ 0.07023	+ 0.02701
4	0.00000	+ 1.32499	+ 0.93180	+ 0.70332	+ 0.56107	+ 0.27456	+ 0.13465	+ 0.05308
5	0.00000	+ 1.15716	+ 1.27695	+ 1.00056	+ 0.81800	+ 0.42405	+ 0.21541	+ 0.08696
6	0.00000	+ 0.73347	+ 1.62516	+ 1.31015	+ 1.09089	+ 0.59231	+ 0.31054	+ 0.12825
7	0.00000	+ 0.39295	+ 1.83406	+ 1.63067	+ 1.37735	+ 0.77589	+ 0.41838	+ 0.17657
8	0.00000	+ 0.18306	+ 1.62896	+ 1.96024	+ 1.67624	+ 0.97245	+ 0.53751	+ 0.23156
9	0.00000	+ 0.07200	+ 1.28838	+ 2.29561	+ 1.98641	+ 1.18047	+ 0.66676	+ 0.29290
10	0.00000	+ 0.02048	+ 0.94535	+ 2.29606	+ 2.30606	+ 1.39901	+ 0.80513	+ 0.36026
15	0.00000	− 0.00311	+ 0.08193	+ 0.88710	+ 2.24321	+ 2.63398	+ 1.61011	+ 0.77799
20	0.00000	− 0.00018	− 0.01273	+ 0.14707	+ 0.88211	+ 4.07566	+ 2.56879	+ 1.30991
25	0.00000	− 0.00001	− 0.00557	− 0.01040	+ 0.21060	+ 5.27421	+ 3.66120	+ 1.93468
30	0.00000	0.00000	− 0.00114	− 0.01752	+ 0.00600	+ 4.53959	+ 4.87954	+ 2.63769
35	0.00000	0.00000	− 0.00018	− 0.00788	− 0.02754	+ 3.18892	+ 6.21442	+ 3.40921
40	0.00000	0.00000	− 0.00002	− 0.00261	− 0.02009	+ 1.97537	+ 7.65013	+ 4.24308
45	0.00000	0.00000	0.00000	− 0.00074	− 0.01015	+ 1.09606	+ 9.16449	+ 5.13551
50	0.00000	0.00000	0.00000	− 0.00019	− 0.00435	+ 0.53331	+10.18760	+ 6.08430
60	0.00000	0.00000	0.00000	− 0.00001	− 0.00061	+ 0.03748	+ 9.05773	+ 8.14613
70	0.00000	0.00000	0.00000	0.00000	− 0.00007	− 0.06409	+ 6.51607	+10.42121
80	0.00000	0.00000	0.00000	0.00000	− 0.00001	− 0.05433	+ 4.12677	+12.90010
90	0.00000	0.00000	0.00000	0.00000	0.00000	− 0.03042	+ 2.34378	+15.56695
100	0.00000	0.00000	0.00000	0.00000	0.00000	− 0.01428	+ 1.17269	+18.39861

3.5 Summary

If the aggregated congestion window size and the aggregated slow start threshold is fairly shared among the TCP connections of a set or an ensemble, EFCM-controlled TCP and standard TCP have nearly the same aggressiveness to the network regarding the aggregated congestion window size and the aggregated slow start threshold after a burst of l consecutive packet losses. A set of n standard TCP connections will have an expected aggregated congestion window size equal or slightly lower than the aggregated congestion window size of n EFCM-controlled TCP connection. In contrast, a set of n standard TCP connections will have an aggregated slow start threshold that is in most cases equal or (slightly) larger than the expected aggregated slow start threshold of n EFCM-controlled TCP connections. The differences in the aggregated congestion window sizes are much lower than the differences in the aggregated slow start thresholds. Therefore, a set of n standard TCP connections is slightly more conservative as an ensemble of n EFCM-controlled TCP connections in sending new TCP segments after a burst of packet loss events, but it is (slightly) more aggressive in sending new TCP segments after the reception of a new TCP acknowledgment.

In general, in a real Internet end system the standard TCP connections of a set do not have equal congestion window sizes and slow start thresholds at the beginning of a packet loss event burst. In this case, the standard TCP connections will also have different probabilities for occurrence of packet loss events in a single TCP connection. Standard TCP connections with a larger congestion window size will have a higher probability for packet loss events than standard TCP connections with a smaller congestion window size. Hence, in the real Internet the aggregated congestion window size of n standard TCP connections will be much smaller than in the considered special case. Therefore, the main reason for the performance gain of the EFCM controller in our simulations is in the built-in fair sharing of the aggregated congestion window size and the aggregated slow start threshold among the n TCP connections of an ensemble.

In addition to the algorithms considered in this chapter, the EFCM controller jointly controls also the smoothed round trip time and the round trip time variance of n TCP connections in an ensemble. With these control algorithms a more adequate value for the current retransmission timer can be calculated. This results in an extra improved performance for the jointly controlled TCP connections of an ensemble compared to standard TCP connections in a set without increasing the aggressiveness to the network of EFCM-controlled TCP connections by violating any standard-conform TCP congestion control algorithms.

Chapter 4

Conclusion

In this technical report, we have analytically shown that an ensemble of n EFCM-controlled TCP connections is nearly as aggressive to the network as a set of n standard TCP connections. For our analysis, we have used the aggregated congestion window size and the aggregated slow start threshold of the set and the ensemble as metrics for the aggressiveness to the network. Hence, we could show that the performance benefits of EFCM are not due to simply increasing aggressiveness, the average number of outstanding packets.

Rather, the analysis indicates that the performance gain of the EFCM approach compared to standard TCP is mainly based on the fairness of the EFCM controller to all TCP connections of an ensemble.

Appendix A

Source code

The tcl-script (a) is used to numerically proof that for two TCP connections the special case with equal congestion window sizes maximizes the expected aggregated congestion window size after l consecutive packet loss events in most cases and maximizes the expected overall aggregated congestion window size in all cases.

The following tcl-scripts are used to numerically determine the values for the aggregated congestion window size and the aggregated slow start threshold (b) for a set of n standard TCP connections and (c) for an ensemble of n EFCM-controlled TCP connections in the general special case, i.e., all TCP connections of the connection set have the same congestion window size and slow start threshold. In addition, the tcl-script in (d) is used to numerically compare the standard TCP and the EFCM congestion control algorithms. The values for the initial and current congestion window size and the initial and current slow start threshold in the tcl-scripts are exemplarily chosen.

(a) standard TCP (2 connections) in a general case

```
##############################
# ttcpc.tcl                  #
# written by Michael Savoric #
#                            #
#       date: 11.12.2002     #
# last update: 13.12.2002    #
##############################

set tcl_precision 17

set cal_mode 0

set dis_mode 0

set l_start 0

if {$dis_mode == 0} {
  set cal_mode 1
}

set packet_loss_probability 0.0001

proc log_fac {n} {
  set sum 0.0
  for {set index 2} {$index <= $n} {incr index} {
    set sum [expr $sum+log($index)]
  }

  return $sum
}

proc probability_for_packet_loss_events {cwnd_1 cwnd_2 k l} {
  set cwnd_agg [expr $cwnd_1+$cwnd_2]

  if {$l > $cwnd_agg || $k > $cwnd_1 || [expr $l-$k] > $cwnd_2} {
    return 0.0
  } else {
    set temp [expr [log_fac $cwnd_1]- \
                   ([log_fac [expr $cwnd_1-$k]]+[log_fac $k])+ \
                   [log_fac $cwnd_2]- \
                   ([log_fac [expr $cwnd_2-($l-$k)]]+[log_fac [expr $l-$k]])- \
                   ([log_fac $cwnd_agg]- \
                    ([log_fac [expr $cwnd_agg-$l]]+[log_fac $l]))]

    return [expr exp($temp)]
  }
}

proc geometrical_distribution {k p} {
  if {$k < 1 || $p > 1} {
    return 0.0
  } else {
    set temp [expr pow(1-$p,$k-1)*$p]

    return $temp
  }
}

proc binomial_distribution {k n p} {
  if {$k > $n || $p > 1} {
    return 0.0
  } else {
    set temp [expr [log_fac $n]-([log_fac [expr $n-$k]]+[log_fac $k])]

    return [expr exp($temp)*pow($p,$k)*pow(1-$p,$n-$k)]
  }
}

for {set cwnd_agg 2} {$cwnd_agg <= 64} {incr cwnd_agg} {
  puts ""

  if {$cal_mode == 0} {
    set p [expr 1.0/$cwnd_agg]
  } else {
    set p $packet_loss_probability
  }

  for {set l $l_start} {$l <= $cwnd_agg} {incr l} {
    puts ""

    for {set cwnd_1 1} {$cwnd_1 <= [expr $cwnd_agg-1]} {incr cwnd_1} {
      set cwnd_2 [expr $cwnd_agg-$cwnd_1]

      set p_l_l [probability_for_packet_loss_events $cwnd_1 $cwnd_2 $l $l]
      set p_0_l [probability_for_packet_loss_events $cwnd_1 $cwnd_2 0 $l]

      set cwnd_agg_after_l_errors($cwnd_1) \
          [expr $p_l_l*(1+$cwnd_2)+$p_0_l*(1+$cwnd_1)+2*(1-$p_l_l-$p_0_l)]
#     puts [format "P_L_L = %8.6lf, P_0_L = %8.6lf, Rest = %8.6lf" \
#                  $p_l_l $p_0_l [expr 1-$p_l_l-$p_0_l]]

      if {$l == $l_start} {
        if {$dis_mode == 0} {
          set sum($cwnd_1) [expr [geometrical_distribution [expr $l+1] [expr 1-$p]]* \
                                 $cwnd_agg_after_l_errors($cwnd_1)]
        } else {
          set sum($cwnd_1) [expr [binomial_distribution $l $cwnd_agg $p]* \
                                 $cwnd_agg_after_l_errors($cwnd_1)]
        }
      } else {
        if {$dis_mode == 0} {
          set sum($cwnd_1) [expr $sum($cwnd_1)+ \
                                 [geometrical_distribution [expr $l+1] [expr 1-$p]]* \
                                 $cwnd_agg_after_l_errors($cwnd_1)]
        } else {
          set sum($cwnd_1) [expr $sum($cwnd_1)+ \
                                 [binomial_distribution $l $cwnd_agg $p]* \
                                 $cwnd_agg_after_l_errors($cwnd_1)]
        }
      }

      if {$cwnd_1 == 1} {
        set maximum($l) $cwnd_agg_after_l_errors($cwnd_1)
      } else {
        if {$cwnd_agg_after_l_errors($cwnd_1) > $maximum($l)} {
          set maximum($l) $cwnd_agg_after_l_errors($cwnd_1)
        }
      }
    }

    for {set cwnd_1 1} {$cwnd_1 <= [expr $cwnd_agg-1]} {incr cwnd_1} {
      puts -nonewline [format "CWND_1 = %3d, CWND_2 = %3d: CWND_AGG(%3d) = %12.8lf " \
                       $cwnd_1 [expr $cwnd_agg-$cwnd_1] $l $cwnd_agg_after_l_errors($cwnd_1)]

      set difference [expr $maximum($l)-$cwnd_agg_after_l_errors($cwnd_1)]

      puts [format "(Difference to the Maximum = %12.8lf)" $difference]
    }
  }

  puts ""

  for {set cwnd_1 1} {$cwnd_1 <= [expr $cwnd_agg-1]} {incr cwnd_1} {
    puts [format "CWND_1 = %3d, CWND_2 = %3d: Mean CWND_AGG = %12.8lf " \
          $cwnd_1 [expr $cwnd_agg-$cwnd_1] $sum($cwnd_1)]
  }
}
```

(b) standard TCP in the special case

```
# tcp_algo_new.tcl
# written by Michael Savoric
#
#        date: 30.10.2002
# last update: 26.11.2002

# number of TCP connection in a virtual ensemble:

set n 3
```

```
# minimum values for cwnd and ssthresh:

set minimum_cwnd 1.0

set minimum_ssthresh [expr $minimum_cwnd+1.0]

# initial and current cwnd:

set initial_cwnd  2.0
set current_cwnd 32.0

if {$current_cwnd < $minimum_cwnd} {
  set current_cwnd $minimum_cwnd
}

# initial and current ssthresh:

set initial_ssthresh 64.0
set current_ssthresh 64.0

if {$current_ssthresh < $minimum_ssthresh} {
  set current_ssthresh $minimum_ssthresh
}

# define helper functions:

proc log_fac {n} {
  set sum 0.0

  for {set index 2} {$index <= $n} {incr index} {
    set sum [expr $sum+log($index)]
  }

  return $sum
}

proc probability_for_packet_loss_events {k l n cwnd} {
  if {$k > $l || $k > $cwnd || $l > ($n > 1 && $l > [expr ($n-1)*$cwnd])} {
    return 0.0
  } else {
    if {$n == 1} {
      if {$k < $l} {
        return 0.0
      } else {
        return 1.0
      }
    } else {
      set temp [expr [log_fac $l]- \
                     ([log_fac [expr $l-$k]]+[log_fac $k])+ \
                     [log_fac [expr $n*$cwnd-$l]]- \
                     ([log_fac [expr ($n-1)*$cwnd-$l+$k]]+[log_fac [expr $cwnd-$k]])- \
                     ([log_fac [expr $n*$cwnd]]- \
                      ([log_fac [expr ($n-1)*$cwnd]]+[log_fac $cwnd]))]

      return exp($temp)
    }
  }
}

# calculate cwnd and ssthresh using the standard TCP control algorithms:

set cwnd $current_cwnd
set cwnd_agg [expr $n*$cwnd]

set current_cwnd_agg $cwnd_agg

set ssthresh $current_ssthresh

set ssthresh_agg [expr $n*$ssthresh]

set current_ssthresh_agg $ssthresh_agg

# number of packet loss events (e.g., retransmission timer timeouts):

set number_of_packet_loss_events $cwnd_agg

set packet_loss_counter 0

puts ""

puts [format "      CWND_AGG = %15.10lf (CWND_AGG reduction-factor = %13.10lf)" \
             $current_cwnd_agg [expr $current_cwnd_agg/$cwnd_agg]]

puts [format "  SSTHRESH_AGG = %15.10lf (CWND_AGG reduction-factor = %13.10lf,\
                               SSTHRESH_AGG reduction-factor = %13.10lf)" \
             $current_ssthresh_agg [expr $current_ssthresh_agg/$cwnd_agg] \
             [expr $current_ssthresh_agg/$ssthresh_agg]]

while {$packet_loss_counter < $number_of_packet_loss_events} {
  incr packet_loss_counter

  set p_0 [probability_for_packet_loss_events 0 $packet_loss_counter $n $cwnd]

  set current_cwnd_agg [expr (1.0-$p_0)*$n*$minimum_cwnd+$p_0*$cwnd_agg]

  set p_1 [probability_for_packet_loss_events 1 $packet_loss_counter $n $cwnd]

  set temp [expr $cwnd/2.0]

  if {$temp < $minimum_ssthresh} {
    set temp $minimum_ssthresh
  }

  set current_ssthresh_agg [expr (1.0-$p_0-$p_1)*$n*$minimum_ssthresh+ \
                                 $p_1*$n*$temp+ \
                                 $p_0*$ssthresh_agg]

  puts ""

  puts [format "packet loss event no. %3d:" $packet_loss_counter]

  puts [format "      CWND_AGG = %15.10lf (CWND_AGG reduction-factor = %13.10lf)" \
               $current_cwnd_agg [expr $current_cwnd_agg/$cwnd_agg]]

  puts [format "  SSTHRESH_AGG = %15.10lf (CWND_AGG reduction-factor = %13.10lf,\
                                 SSTHRESH_AGG reduction-factor = %13.10lf)" \
               $current_ssthresh_agg [expr $current_ssthresh_agg/$cwnd_agg] \
               [expr $current_ssthresh_agg/$ssthresh_agg]]
}
```

(c) EFCM-controlled TCP

```
# efcm_algo.tcl
# written by Michael Savoric
```

```
#
#        date: 29.10.2002
# last update: 19.11.2002

# number of TCP connection in an ensemble:

set n 3

# minimum values for cwnd and ssthresh:

set minimum_cwnd 1.0

set minimum_ssthresh [expr $minimum_cwnd+1.0]

# initial and current cwnd:

set initial_cwnd  2.0
set current_cwnd 32.0

if {$current_cwnd < $minimum_cwnd} {
  set current_cwnd $minimum_cwnd
}

# initial and current ssthresh:

set initial_ssthresh 64.0
set current_ssthresh 64.0

if {$current_ssthresh < $minimum_ssthresh} {
  set current_ssthresh $minimum_ssthresh
}

# calculate cwnd and ssthresh using the EFCM control algorithms:

set cwnd $current_cwnd

set cwnd_agg [expr $n*$cwnd]

set current_cwnd_agg $cwnd_agg

set ssthresh $current_ssthresh

set ssthresh_agg [expr $n*$ssthresh]

set current_ssthresh_agg $ssthresh_agg

# number of packet loss events (e.g., retransmission timer timeouts):

set number_of_packet_loss_events $cwnd_agg

set packet_loss_counter 0

puts ""

puts [format "      CWND_AGG = %11.6lf (CWND_AGG reduction-factor = %9.6lf)" \
             $current_cwnd_agg [expr $current_cwnd_agg/$cwnd_agg]]

puts [format "  SSTHRESH_AGG = %11.6lf (CWND_AGG reduction-factor = %9.6lf,\
                                   SSTHRESH_AGG reduction-factor = %9.6lf)" \
             $current_ssthresh_agg [expr $current_ssthresh_agg/$cwnd_agg] \
             [expr $current_ssthresh_agg/$ssthresh_agg]]

while {$packet_loss_counter < $number_of_packet_loss_events} {
  incr packet_loss_counter
  set temp [expr $cwnd/2.0]

  if {$temp < $minimum_ssthresh} {
    set temp $minimum_ssthresh
  }

  set current_ssthresh_agg [expr $current_ssthresh_agg-$ssthresh+$temp]

  set ssthresh [expr $current_ssthresh_agg/$n]

  if {$ssthresh < $minimum_ssthresh} {
    set current_ssthresh_agg [expr $current_ssthresh_agg+$n*($minimum_ssthresh-$ssthresh)]

    set ssthresh $minimum_ssthresh
  }

  set current_cwnd_agg [expr $current_cwnd_agg-($cwnd-$minimum_cwnd)]

  set cwnd [expr $current_cwnd_agg/$n]

  if {$cwnd < $minimum_cwnd} {
    set current_cwnd_agg [expr $current_cwnd_agg+$n*($minimum_cwnd-$cwnd)]

    set $cwnd $minimum_cwnd
  }

  puts ""

  puts [format "packet loss event no. %3d:" $packet_loss_counter]

  puts [format "      CWND_AGG = %11.6lf (CWND_AGG reduction-factor = %9.6lf)" \
               $current_cwnd_agg [expr $current_cwnd_agg/$cwnd_agg]]

  puts [format "  SSTHRESH_AGG = %11.6lf (CWND_AGG reduction-factor = %9.6lf,\
                                 SSTHRESH_AGG reduction-factor = %9.6lf)" \
               $current_ssthresh_agg [expr $current_ssthresh_agg/$cwnd_agg] \
               [expr $current_ssthresh_agg/$ssthresh_agg]]
}
```

(d) Comparison of standard TCP in the special case with EFCM-controlled TCP

```
# tcp_efcm_delta_new.tcl
# written by Michael Savoric
#
#        date: 20.11.2002
# last update: 26.11.2002

set tcl_precision 17

# number of TCP connection in an ensemble:

set n 3

# minimum values for cwnd and ssthresh:

set minimum_cwnd 1.0

set minimum_ssthresh [expr $minimum_cwnd+1.0]
```

```
# initial and current cwnd:

set initial_cwnd   2.0
set current_cwnd  32.0

if {$current_cwnd < $minimum_cwnd} {
  set current_cwnd $minimum_cwnd
}

# initial and current ssthresh:

set initial_ssthresh 64.0
set current_ssthresh 64.0

if {$current_ssthresh < $minimum_ssthresh} {
  set current_ssthresh $minimum_ssthresh
}

# define helper functions:

proc log_fac {n} {
  set sum 0.0

  for {set index 2} {$index <= $n} {incr index} {
    set sum [expr $sum+log($index)]
  }

  return $sum
}

proc probability_for_packet_loss_events {k l n cwnd} {
  if {$k > $cwnd || $k > $l || ($n > 1 && $l > [expr ($n-1)*$cwnd])} {
    return 0.0
  } else {
    if {$n == 1} {
      if {$k < $l} {
        return 0.0
      } else {
        return 1.0
      }
    } else {
      set temp [expr [log_fac $l]- \
                     ([log_fac [expr $l-$k]]+[log_fac $k])+ \
                     [log_fac [expr $n*$cwnd-$l]]- \
                     ([log_fac [expr ($n-1)*$cwnd-$l+$k]]+[log_fac [expr $cwnd-$k]])- \
                     ([log_fac [expr $n*$cwnd]]- \
                      ([log_fac [expr ($n-1)*$cwnd]]+[log_fac $cwnd]))]

      return exp($temp)
    }
  }
}

# calculate cwnd and ssthresh using the standard and the EFCM control algorithms:

set cwnd_tcp $current_cwnd

set cwnd_agg_tcp [expr $n*$cwnd_tcp]

set current_cwnd_agg_tcp $cwnd_agg_tcp

set ssthresh_tcp $current_ssthresh

set ssthresh_agg_tcp [expr $n*$ssthresh_tcp]
set current_ssthresh_agg_tcp $ssthresh_agg_tcp

set cwnd_efcm $current_cwnd

#

set cwnd_agg_efcm [expr $n*$cwnd_efcm]

set current_cwnd_agg_efcm $cwnd_agg_efcm

set ssthresh_efcm $current_ssthresh

set ssthresh_agg_efcm [expr $n*$ssthresh_efcm]

set current_ssthresh_agg_efcm $ssthresh_agg_efcm

# number of packet loss events (e.g., retransmission timer timeouts):

set number_of_packet_loss_events $cwnd_agg_tcp

set packet_loss_counter 0

puts ""

puts [format "       CWND_AGG (TCP) = %15.10lf  (CWND_AGG (TCP) reduction-factor = %13.10lf)" \
             $current_cwnd_agg_tcp [expr $current_cwnd_agg_tcp/$cwnd_agg_tcp]]

puts [format "   SSTHRESH_AGG (TCP) = %15.10lf  (CWND_AGG (TCP) reduction-factor = %13.10lf, \
                                   SSTHRESH_AGG (TCP) reduction-factor = %13.10lf)" \
             $current_ssthresh_agg_tcp [expr $current_ssthresh_agg_tcp/$cwnd_agg_tcp] \
             [expr $current_ssthresh_agg_tcp/$ssthresh_agg_tcp]]

puts [format "      CWND_AGG (EFCM) = %15.10lf (CWND_AGG (EFCM) reduction-factor = %13.10lf)" \
             $current_cwnd_agg_efcm [expr $current_cwnd_agg_efcm/$cwnd_agg_efcm]]

puts [format "  SSTHRESH_AGG (EFCM) = %15.10lf (CWND_AGG (EFCM) reduction-factor = %13.10lf,\
                                   SSTHRESH_AGG (EFCM) reduction-factor = %13.10lf)" \
             $current_ssthresh_agg_efcm [expr $current_ssthresh_agg_efcm/$cwnd_agg_efcm] \
             [expr $current_ssthresh_agg_efcm/$ssthresh_agg_efcm]]

while {$packet_loss_counter < $number_of_packet_loss_events} {
  incr packet_loss_counter

  set p_0 [probability_for_packet_loss_events 0 $packet_loss_counter $n $cwnd_tcp]

  set current_cwnd_agg_tcp [expr (1.0-$p_0)*$n*$minimum_cwnd+$p_0*$cwnd_agg_tcp]

  set p_1 [probability_for_packet_loss_events 1 $packet_loss_counter $n $cwnd_tcp]

  set temp_tcp [expr $cwnd_tcp/2.0]

  if {$temp_tcp < $minimum_ssthresh} {
    set temp_tcp $minimum_ssthresh
  }

  set current_ssthresh_agg_tcp [expr (1.0-$p_0-$p_1)*$n*$minimum_ssthresh+ \
                                     $p_1*$n*$temp_tcp+ \
                                     $p_0*$ssthresh_agg_tcp]

  puts ""

  set temp_efcm [expr $cwnd_efcm/2.0]
```

```
if {$temp_efcm < $minimum_ssthresh} {
  set temp_efcm $minimum_ssthresh
}

set current_ssthresh_agg_efcm [expr $current_ssthresh_agg_efcm-$ssthresh_efcm+$temp_efcm]

set ssthresh_efcm [expr $current_ssthresh_agg_efcm/$n]

if {$ssthresh_efcm < $minimum_ssthresh} {
  set current_ssthresh_agg_efcm \
      [expr $current_ssthresh_agg_efcm+$n*($minimum_ssthresh-$ssthresh_efcm)]

  set ssthresh_efcm $minimum_ssthresh
}

set current_cwnd_agg_efcm [expr $current_cwnd_agg_efcm-($cwnd_efcm-$minimum_cwnd)]

set cwnd_efcm [expr $current_cwnd_agg_efcm/$n]

if {$cwnd_efcm < $minimum_cwnd} {
  set current_cwnd_agg_efcm [expr $current_cwnd_agg_efcm+$n*($minimum_cwnd-$cwnd_efcm)]

  set $cwnd_efcm $minimum_cwnd
}

puts ""

puts [format "packet loss event no. %4d:" $packet_loss_counter]

set cwnd_rf_tcp [expr $current_cwnd_agg_tcp/$cwnd_agg_tcp]
set ssthresh_cwnd_rf_tcp [expr $current_ssthresh_agg_tcp/$cwnd_agg_tcp]
set ssthresh_rf_tcp [expr $current_ssthresh_agg_tcp/$ssthresh_agg_tcp]

puts [format "       CWND_AGG (TCP) = %15.10lf  \
             (CWND_AGG (TCP) reduction-factor = %13.10lf)" \
             $current_cwnd_agg_tcp $cwnd_rf_tcp]

puts [format "   SSTHRESH_AGG (TCP) = %15.10lf  \
             (CWND_AGG (TCP) reduction-factor = %13.10lf, \
             SSTHRESH_AGG (TCP) reduction-factor = %13.10lf)" \
             $current_ssthresh_agg_tcp $ssthresh_cwnd_rf_tcp $ssthresh_rf_tcp]

set cwnd_rf_efcm [expr $current_cwnd_agg_efcm/$cwnd_agg_efcm]
set ssthresh_cwnd_rf_efcm [expr $current_ssthresh_agg_efcm/$cwnd_agg_efcm]
set ssthresh_rf_efcm [expr $current_ssthresh_agg_efcm/$ssthresh_agg_efcm]

puts [format "      CWND_AGG (EFCM) = %15.10lf \
             (CWND_AGG (EFCM) reduction-factor = %13.10lf)" \
             $current_cwnd_agg_efcm $cwnd_rf_efcm]

puts [format "  SSTHRESH_AGG (EFCM) = %15.10lf \
             (CWND_AGG (EFCM) reduction-factor = %13.10lf,\
             SSTHRESH_AGG (EFCM) reduction-factor = %13.10lf)" \
             $current_ssthresh_agg_efcm $ssthresh_cwnd_rf_efcm $ssthresh_rf_efcm]

puts ""

set delta_cwnd_agg [expr $current_cwnd_agg_tcp-$current_cwnd_agg_efcm]
set delta_cwnd_rf [expr $cwnd_rf_tcp-$cwnd_rf_efcm]
set delta_ssthresh_agg [expr $current_ssthresh_agg_tcp-$current_ssthresh_agg_efcm]
set delta_ssthresh_cwnd_rf [expr $ssthresh_cwnd_rf_tcp-$ssthresh_cwnd_rf_efcm]
set delta_ssthresh_rf [expr $ssthresh_rf_tcp-$ssthresh_rf_efcm]

puts [format "       Delta CWND_AGG = %15.10lf  \
               (Delta CWND_AGG reduction-factor = %13.10lf)" \
               $delta_cwnd_agg $delta_cwnd_rf]

  puts [format "   Delta SSTHRESH_AGG = %15.10lf  \
               (Delta CWND_AGG reduction-factor = %13.10lf, \
               Delta SSTHRESH_AGG reduction-factor = %13.10lf)" \
               $delta_ssthresh_agg $delta_ssthresh_cwnd_rf $delta_ssthresh_rf]

  if {$packet_loss_counter == 1} {
    set max_cwnd_packet_loss_counter 1
    set max_delta_cwnd_agg $delta_cwnd_agg

    set min_cwnd_packet_loss_counter 1
    set min_delta_cwnd_agg $delta_cwnd_agg

    set max_ssthresh_packet_loss_counter 1
    set max_delta_ssthresh_agg $delta_ssthresh_agg

    set min_ssthresh_packet_loss_counter 1
    set min_delta_ssthresh_agg $delta_ssthresh_agg
  } else {
    if {$delta_cwnd_agg < $min_delta_cwnd_agg} {
      set min_cwnd_packet_loss_counter $packet_loss_counter
      set min_delta_cwnd_agg $delta_cwnd_agg
    }

    if {$delta_cwnd_agg > $max_delta_cwnd_agg} {
      set max_cwnd_packet_loss_counter $packet_loss_counter
      set max_delta_cwnd_agg $delta_cwnd_agg
    }

    if {$delta_ssthresh_agg < $min_delta_ssthresh_agg} {
      set min_ssthresh_packet_loss_counter $packet_loss_counter
      set min_delta_ssthresh_agg $delta_ssthresh_agg
    }

    if {$delta_ssthresh_agg > $max_delta_ssthresh_agg} {
      set max_ssthresh_packet_loss_counter $packet_loss_counter
      set max_delta_ssthresh_agg $delta_ssthresh_agg
    }
  }
}

puts ""

puts [format "    MAX Delta CWND_AGG = %15.10lf     \
             (MAX Delta CWND_AGG reduction factor = %13.10lf) \[%4d\]"\
             $max_delta_cwnd_agg [expr $max_delta_cwnd_agg/$cwnd_agg_tcp] \
             $max_cwnd_packet_loss_counter]

puts [format "    MIN Delta CWND_AGG = %15.10lf     \
             (MIN Delta CWND_AGG reduction factor = %13.10lf) \[%4d\]"\
             $min_delta_cwnd_agg [expr $min_delta_cwnd_agg/$cwnd_agg_tcp] \
             $min_cwnd_packet_loss_counter]

puts [format "MAX Delta SSTHRESH_AGG = %15.10lf \
             (MAX Delta SSTHRESH_AGG reduction factor = %13.10lf) \[%4d\]"\
             $max_delta_ssthresh_agg [expr $max_delta_ssthresh_agg/$ssthresh_agg_tcp] \
             $max_ssthresh_packet_loss_counter]

puts [format "MIN Delta SSTHRESH_AGG = %15.10lf \
             (MIN Delta SSTHRESH_AGG reduction factor = %13.10lf) \[%4d\]"\
             $min_delta_ssthresh_agg [expr $min_delta_ssthresh_agg/$ssthresh_agg_tcp] \
             $min_ssthresh_packet_loss_counter]
```

Acknowledges

The work presented in this technical report has been partially supported by a research contract with Ericsson Eurolab, Aachen, Germany.

Bibliography

[1] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[2] L. Eggert, T. Henderson, and J. Touch. Effects of ensemble TCP. *ACM SIGCOMM Computer Communication Review*, 30(January):15–29, 2000.

[3] S. Floyd and T. Henderson. The new reno modification to TCP´s fast recovery algorithm. RFC 2582, 1999.

[4] M. Savorić. Identifying and evaluating the potential of reusing network information from different flows. Technical report, TKN-01-019, http://www-tkn.ee.tu-berlin.de/publications/papers/ccc_tr.pdf, 2001.

[5] M. Savorić. The TCP control block interdependence in fixed networks — some performance results. In *Proceedings QOFIS 2001*, LNCS 2156, pages 261–272, 2001.

[6] M. Savorić and H. Karl. Performance evaluation of an improved common congestion controller for TCP connections — new simulation results. Technical report, TKN, http://www-tkn.ee.tu-berlin.de/publications/papers/efcm_tr_3.pdf, 2003.

[7] M. Savorić, H. Karl, and A. Wolisz. The TCP control block interdependence in fixed networks — new performance results. *to be published in Computer Communications*, 2002.

[8] J. Touch. TCP control block interdependence. RFC 2140, 1997.