Problems and Improvements of Internet Traffic Congestion Control

Master's Thesis completed at Control & Communication,
Linköpings Universitet, Sweden
by

Frida Gunnarsson

Reg nr: LiTH-ISY-EX-3098

Problems and Improvements of Internet Traffic Congestion Control

Master's Thesis completed at Control & Communication,
Linköpings Universitet, Sweden
by

Frida Gunnarsson

Reg nr: LiTH-ISY-EX-3098

Supervisor: **Fredrik Gunnarsson**

Examiner: **Fredrik Gustafsson**

Linköping, 17th October 2000.

Abstract

The main protocol for flow and congestion control on the Internet is the Transfer Control Protocol, TCP. This protocol was constructed and developed based on heuristic arguments, and its main purpose is to prevent network congestion. Because of shifts in the Internet traffic, TCP does not work as well as when it was designed - a problem that has been addressed by researchers in different ways. This report gives an overview of the different TCP performance aspects and active areas of research. The research directions address different problem areas and there are two major proposals concerning congestion control and TCP. They discuss the use of explicit congestion notification and of automatic control theory to enhance the performance of the existing congestion control in TCP.

A proposed scheme with feedback control and a Smith predictor is discussed and compared with ordinary TCP, in a simulation study. The simulations have been carried out for a single TCP connection with a model implemented in Simulink and Stateflow. The report shows that the feedback solution improves the performance of TCP. With feedback TCP and for this single connection, we manage to avoid packet losses, reduce the time delay and stabilize the traffic load.

Key Words: TCP, transfer control protocol, congestion control, Internet, Smith predictor, time delay system, Simulink, StateFlow

To Anders

Contents

List of Figures

List of Tables

Notation

Variables & Symbols

AdvWnd,ssthresh, ...	variables in TCP
StoreAndSend,*AckNo.*, ...	SIMULINK or STATEFLOW blocks
Main/, *Arrival/*, ...	internal states in a STATEFLOW block.
MaxSend, *UpStart*, ...	events or data transmitted in SIMULINK and STATEFLOW.

Abbreviations

ABR	Available Bit Rate
ack	acknowledgment
AdvWnd	Advertised Window
AIMD	Additive Increase Multiplicative Decrease
ATM	Asynchronous Transfer Mode
BCN	Binary Congestion Notification
BW	Bandwidth
cwnd	congestion window
ECN	Explicit Congestion Notification

ER	Explicit Rate
ERN	Explicit Rate Notification
GAW	Generalized Advertised Window
HPF	Heterogenous Packet Flows
IP	Internet Protocol
RAP	Rate Adaption Protocol
RED	Random Early Detection
RTT	Round Trip Time
ssthresh	slow start threshold
TCP	Transfer Control Protocol
TOL	TimeOut Limit

Chapter 1

Introduction

The use for automatic control methods in communication systems is a hot topic in this information era. Being one of the largest man-made communication systems in the world, the Internet with its control algorithms is an interesting and challenging area of research for people in automatic control. The Internet is in constant change. The number of users is increasing and there has been a shift in the traffic characteristics: more traffic and shorter transmissions.

The most widely used control algorithm is the one in the transfer control protocol, TCP. Its purpose is to control the flow and prevent congestion. TCP was developed during the birth of Internet, based on heuristic arguments and the current traffic situation. The fast growth of the Internet has changed the conditions that TCP was made for. The challenge for researchers is to come up with significant improvements that require as few changes as possible. Backward compatibility is also an issue here.

The purpose of this report is to summarize the existing research on Internet flow and congestion control, especially the different directions concerning TCP. It is also to investigate the usefulness of automatic control on the Internet, and to be a base and inspiration for continued work. The purpose of the work that resulted in this report was also to make a simulation environment that illustrates the behavior of TCP.

This report focuses on TCP and on the research around this protocol.

We will start in Chapter 2 by explaining how TCP works and exemplifying some of its behavior, using data from simulations. Here we will also discuss some of the recent research on TCP congestion control. In Chapter 3 we will concentrate on methods that have been developed using a control theoretic framework, we will present the theory and provide some results from simulations. These results will be qualitatively compared with the results from simulations using ordinary TCP. Finally, in Chapter 4 we will summarize our results, give some conclusions and discuss future work.

The simulation model is presented in Appendix A. Appendix B provides a list of articles sorted by the area of research that they address. Along with every article is also a short description of their content and results. This part of the report is intended for people with an interest in the whole picture of Internet traffic flow control research.

This report is the final exam for a Master's program at the Linköpings Universitet, Sweden. It is the completion of a Master of Science degree in Applied Physics and Electrical Engineering. The report was written at the Division of Control & Communication, Department of Electrical Engineering, Linköpings Universitet. It was written, using LaTeX 2_ε, for everyone with an interest in Internet technology, but assumes that the reader has some basic knowledge in mathematics, transform theory, programming and control engineering.

Chapter 2

Transfer Control Protocol, TCP

This chapter will give a short description on how TCP operates to avoid congestion and also point out some of the problems with its behavior in today's Internet. We will also use simulated data to exemplify the behavior of TCP. For a more thorough description on TCP, please refer to Stevens [33].

2.1 TCP and Congestion

TCP is the main transfer protocol on the Internet today. Its purpose is to control the flow of data and to prevent the network from getting congested. In the following discussion the end computers will be called hosts and the computers along the path between them will be called routers, as in route or path. When a host wants to send data to another host it has to establish a TCP connection. TCP divides the data into segments. The two hosts together decides the size of these segments in the preliminary phase of the connection. The preliminary phase is often called the three-way handshake, because it consists of three transmissions between the two hosts. We will not discuss how this preliminary phase works in this report. When the handshake is complete the hosts can start sending data. The data segments will be referred to as *packets* in the following discussion.

The TCP supplies a *header* to every packet. This header contains information about the sender and the packet, to be used by the receiver. A

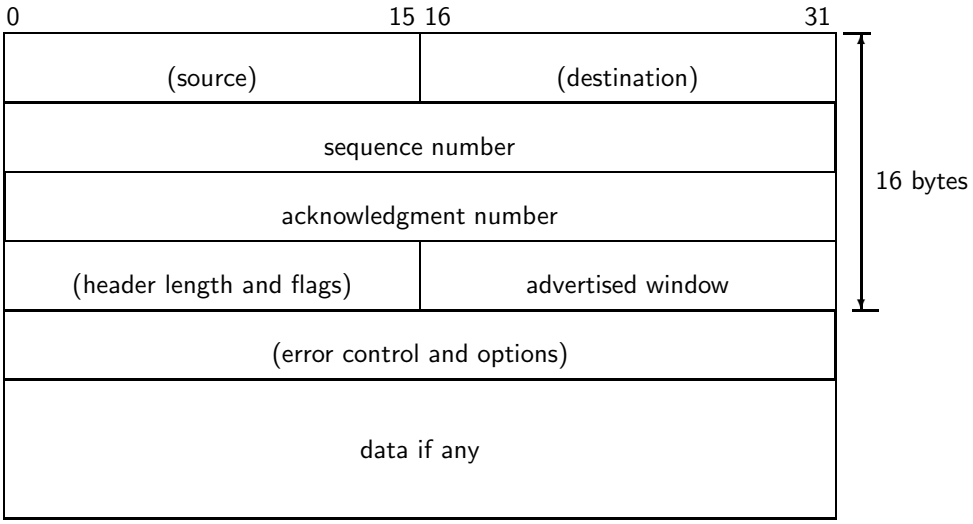

Figure 2.1. *A simplified picture of the TCP header. The fieldnames in brackets will not be explained nor discussed here.*

simplified version of the header is given in Figure 2.1. Some of the fields have their field names in brackets, and they will not be discussed here. Every packet is given a *sequence number* since we have to know the correct order of the packets. Among the information in the header, we will focus on the *advertised window* and the *acknowledgment number*. All connections on the Internet are bidirectional, i.e. the roles of sender and receiver are shared between both end hosts. For simplicity we will discuss only the functions in one direction. In our discussion the *sender* is the host sending data packets and the *receiver* is the host collecting them and responding to them. The *receiver* has an internal buffer in which the data packets are stored and processed. We will denote the size of this buffer with ReceiversBuffer.

We are now going to discuss the information in the TCP header that we are interested in. As mentioned, the sequence number indicates the position of the packet in the original data. The Advertised Window, AdvWnd, is a measure of how much the *receiver* is capable of receiving. If we denote the last byte received by the *receiver* with LastRcvByte and the next byte the application at the *receiver* will process with NextByteToRead we get

$$\mathsf{AdvWnd} = \mathsf{ReceiversBuffer} - (\mathsf{LastRcvByte} - \mathsf{NextByteToRead}),$$

i.e. how much space there is left in the *receiver*'s buffer.

The Acknowledgment Number, ack no., is used by the *receiver* as a receipt of received data; this field indicates the sequence number of the next packet that the *receiver* expects. The *receiver* will acknowledge every packet received, if a packet is lost the *receiver* will send duplicate acks for this packet as out-of-order packets arrive. E.g. the incoming packets arrive in the following sequence: 1,2,4,5. The corresponding sequence of acknowledgments will then be: 2,3,3,3, because the *receiver* is still waiting for packet number three to arrive. The "early" packets are stored until the lost packet is received, and then they are all passed to the buffer in the right order. We will assume that every packet is acknowledged with a separate ack. This is not true in the real implementations of the TCP, but it simplifies the theoretic presentation. In reality one ack can acknowledge more than one arrived packet. The changes in the following discussion are straightforward.

For simplicity we now introduce

$$\mathsf{PacketsOut} = \mathsf{LastSentPacket} - \mathsf{LastAckedPacket},$$

for the packets that have been sent but no corresponding ack has arrived yet, also called the *outstanding packets*.

Another limit on the amount of data that can be sent is the capacity of the network. The capacity is measured in how much data the network can carry and process at each time instant, measured in bytes per second. When a TCP connection is initialized a share of the capacity is assigned to this connection, this share is called the *available bandwidth* for the connection. The size of the available bandwidth depends on the load and the total bandwidth of the network. The size is also unknown to the connection. If there are many connections at the same time the bandwidth for each connection will be small and vice versa. If the number of connections varies then the size of the available bandwidth varies during a connection.

The *sender* receives information about the capacity of the *receiver* in incoming packets, but the network has no way of telling the *sender* about its capacity. Therefore, every host has an internal window called the congestion window, cwnd. The cwnd is used to limit the amount of data sent, based on the state of the network. The cwnd is a way for the host to estimate the available bandwidth. As long as every packet is delivered the value of the cwnd will be increased and when a packet is lost its value will be decreased. We will later explain exactly how the size of the cwnd is set. In this discussion we are only interested in the *sender*'s cwnd. Thus, the cwnd

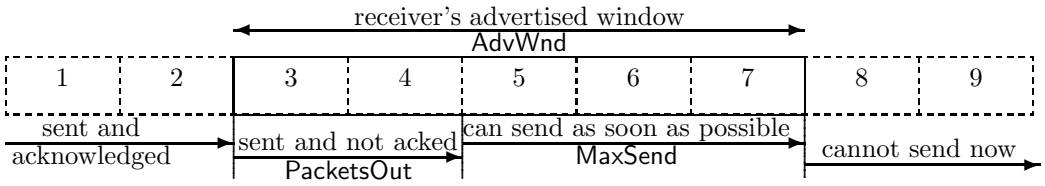

Figure 2.2. *TCP's sliding window. The* sender *has transmitted four sequences and received acknowledgments for two.*

is an estimate of the network capacity and the AdvWnd denotes capacity of the *receiver*. The *sender*'s cwnd and the *receiver*'s AdvWnd together decide how much data the *sender* can transmit at each time instant. We denote the maximum amount of data that is allowed to send with MaxSend.

$$\mathsf{MaxSend} = \min(\mathsf{cwnd}, \mathsf{AdvWnd}) - \mathsf{PacketsOut} \tag{2.1}$$

This method of deciding how much data to send is called a sliding window. TCP's sliding window is visualized in Figure 2.2, we assume here that the cwnd is larger than the AdvWnd. If this was not the case, the size of the cwnd would be the limiting factor instead of the AdvWnd in Figure 2.2. The window will move, shrink and grow during a transmission, that is why it is called a sliding window. Note that the leftmost side of the window can never move to the left.

As mentioned before the size of the cwnd is dynamic and TCP uses it to avoid congestion. Every time a non-duplicate ack is received by the *sender*, the size of the cwnd is changed according to predefined rules, which we will describe in the following part of this section.

There are two phases for the connection, *Slow start* and *Congestion avoidance*, and different updating rules apply for the two phases. The Slow start phase is used to quickly reach an appropriate send rate starting from minimum. Then the Congestion avoidance phase takes over and slowly increases the send rate. To separate the two phases the *sender* uses another internal parameter called the ssthresh (slow start threshold). In our discussion the values of cwnd and ssthresh has the unit of measurement "number of packets". The two phases are separated as follows:

```
if cwnd <= ssthresh
    slow start
else
    congestion avoidance
endif
```

When the phase is determined the *sender* can update the internal parameters accordingly. This happens, as mentioned before, every time a non-duplicate ack arrives.

```
if new connection
    cwnd = one packet
elseif slow start
    cwnd = cwnd + one
elseif congestion avoidance
    cwnd = cwnd + 1/cwnd
endif
```

Assume that MaxSend packets are sent at every time unit and the corresponding acks arrive after exactly one time unit. Also assume that the AdvWnd is larger than the cwnd. Then, let $c(n)$ be the value of the cwnd at time n, $p(n)$ be the number of packets sent at time n and s be the value of the ssthresh. $p(n-1)$ acks arrive at time n and the value of $c(n)$ is therefore updated $p(n-1)$ times. We can then express the updating rules as follows

$$c(n) = \begin{cases} 1 & \text{if } n = 0 \\ c(n-1) + p(n-1) & \text{if } c(n-1) \leq s \\ c(n-1) + p(n-1)/c(n-1) & \text{otherwise} \end{cases} \tag{2.2}$$

In this ideal case $p(n) = c(n)$, which means we get

$$c(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2c(n-1) & \text{if } c(n-1) \leq s \\ c(n-1) + 1 & \text{otherwise} \end{cases} \tag{2.3}$$

this gives us

$$c(n) = \begin{cases} 2^n & \text{for } n \leq N \\ c(N) + (n - N) & \text{otherwise} \end{cases} \tag{2.4}$$

where the constant N is given by the relation $2^{N-1} \leq s < 2^N$.

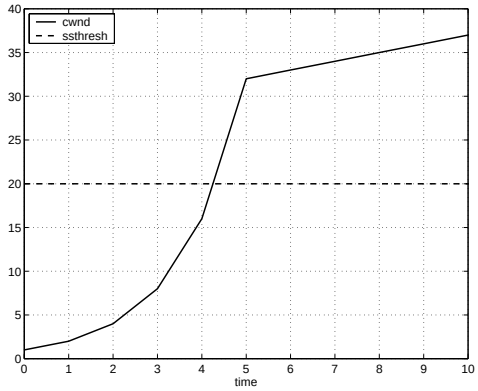

Figure 2.3. *The ideal increase of the congestion window. At the beginning of each time step, a burst of allowed packets are sent and they all return after one time unit, which results in an increase of the* cwnd.

This gives us an ideal increase of the cwnd as shown i Figure 2.3. At time zero one packet is sent and the corresponding ack arrives at time one. This sets the cwnd to two and two packets are sent. Their acks arrive at time three and the cwnd is set to four etc. In this case $N = 5$.

This increase of the cwnd will eventually result in a packet loss, i.e. the amount of sent packets will overflow one of the routers along the way and the arriving packets will be lost. Packets might also get lost because of transmission errors, routing path errors or something else. Though, TCP assumes that a packet loss means that the system is congested and therefore decides that something has to be done. There are two ways for the *sender* to find out about a packet loss:

Three duplicate acks. When three duplicate acks are received, the *sender* retransmits the missing packet and updates the internal parameters. (The third duplicate ack will be the forth ack with the same number.) Remember that the ack tells the *sender* which packet the *receiver* is expecting next.

Timeout. When no ack has been received before the timeout limit, TOL, has expired, the *sender* retransmits all packets that have been sent since the presumed loss.

The actions on congestion are to recalculate the cwnd and the ssthresh, and thereby change the rate of the outgoing data. The recalculations are made based on the following assumptions:

- A timeout indicates that the connection has been idle for a longer period and that this is due to a major congestion, therefore the cwnd is reset to its initial value.
- The three duplicate acks indicates a single packet loss and, thus, the cwnd is not decreased as dramatically as after a timeout.

The following rules are used

```
if three duplicate acks
    cwnd = ssthresh + 3
    ssthresh = max[0.5 * PacketsOut, 2]
    when lost packet acked
        cwnd = ssthresh
elseif timeout
    ssthresh = max[0.5 * PacketsOut, 2]
    cwnd = one
endif
```

To determine the appropriate timeout limit, the *sender* has to know how much time the system needs to deliver a packet and the corresponding ack. This time is called the round trip time, RTT, and the sender calculates an average RTT estimate, $R(t)$, based on measurements, $M(t)$, and stores it as an internal parameter. The estimate, $R(t)$, and the timeout limit, $TOL(t)$, can be calculated using the low pass filter

$$R(t_{new}) = \alpha R(t_{old}) + (1-\alpha)M(t_{new})$$
$$TOL(t) = \max(\min(\beta R(t), ub), lb)$$

where ub is an upper bound on the timeout (e.g. 1 minute) and lb is a lower bound on the timeout (e.g. 1 second). α is a smoothing factor (e.g. 0.8 to 0.9) and β is a delay variance factor (e.g. 1.3 to 2.0). The times for the update depend on sending and incoming times. A specific packet was sent between times t_{old} and t_{new}, and the corresponding ack arrives at time t_{new}, which gives a new measurement, $M(t_{new})$ (the last measurement we got was $M(t_{old})$). The filter is a discrete event clocked low pass filter. The parameter α tunes the tradeoff between fast adaptivity to changes in the RTT and noise reduction. More recent TCP-versions also use an estimate of the variance of $R(t)$ to get a better value on the timeout limit.

The changes in the cwnd at a timeout and after three acks are shown in Figure 2.4. The sent sequence number, the received acknowledgment

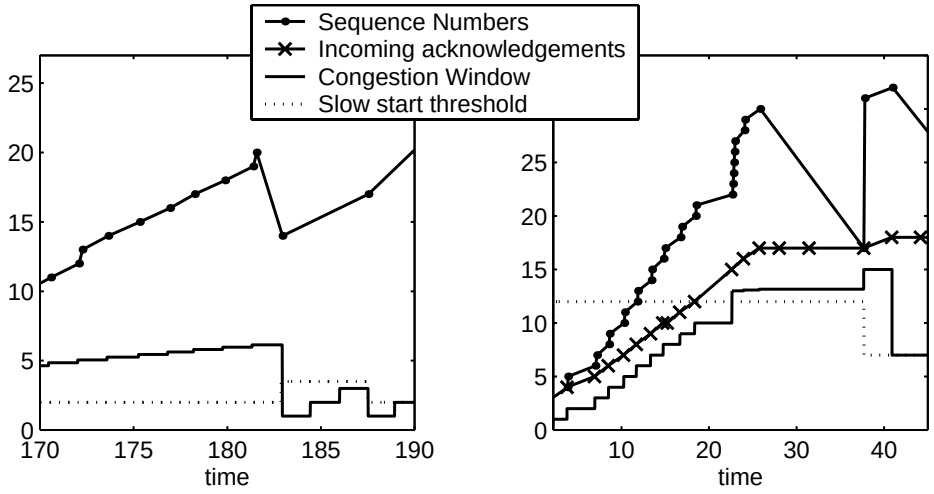

Figure 2.4. *The cwnd upon congestion. The value changes in different ways for the two different congestion cases. Timeout to the left and three duplicate acks to the right.*

number, the cwnd and the ssthresh are shown. In the leftmost figure we see that TCP gets an indication of timeout at time 183. This sets the cwnd to one and the ssthresh is set to half of the number of outstanding packets $(0.5 * (20 - 13) = 3.5)$. The cwnd is then increased with one every time a new ack arrives until, in this case, a new timeout is detected. In the rightmost figure we see that the third duplicate ack for sequence number 17 arrives at time 38. The cwnd is then set to the ssthresh plus three $(12 + 3 = 15)$ and the ssthresh is once again set to half of the outstanding packets $(0.5 * (30 - 16) = 7)$. When a new ack arrives the cwnd is set to the value of ssthresh (7) and then ordinary congestion control resumes.

These rules for the TCP congestion control are Internet Standard, and can be found in e.g. requests for comments [2].

We are interested in applications of automatic control methods on communication systems. As a start, Table 2.1 is a summary of the parameters that concern the TCP congestion control algorithm.

Table 2.1. *Parameters in the TCP control algorithm*

Internal parameters:	cwnd, ssthresh, RTT-estimate
For the header:	ack.no., seq.no., AdvWnd
Measurements:	number of duplicate acks, RTT
Unknown:	available bandwidth, BW
System/network characteristics:	RTT (varies), total bandwidth.
Controls:	output flow

2.2 Problems With TCP

A lot has changed since TCP was invented. Here follows some of the problems that have been addressed by different researchers. In Section 2.4 and, more thoroughly, in Appendix B some of the work around these problems is summarized.

1. TCP provokes packet losses to get an idea about the state of the network. The protocol has no explicit feedback. We get implicit feedback when something goes wrong.
2. The use of the available bandwidth oscillates. This means that the load of the network varies;
   small load: space left, unused available capacity,
   overload: network overflow, packet losses, delays, retransmissions, i.e. slower than it has to be.
3. There has been a shift in the Internet traffic. Due to short transmissions, TCP is "always" in slowstart, measurements in [5] show 85% in slowstart.
4. There is no difference between different traffic classes, such as ftp files, video streams or e-mail.
5. Originally less than 1% packet losses, now $5-7\%$, which means a lot of retransmissions, (13%), most of which are due to timeouts. (The measurements can be found in [5].)
6. TCP always assumes that packet losses are due to congestion.

2.3 The behavior of TCP - Modeling and Measurements

This section shows some examples on TCP's behavior in different situations. We will use the problems described in the previous section as a base for our discussion.

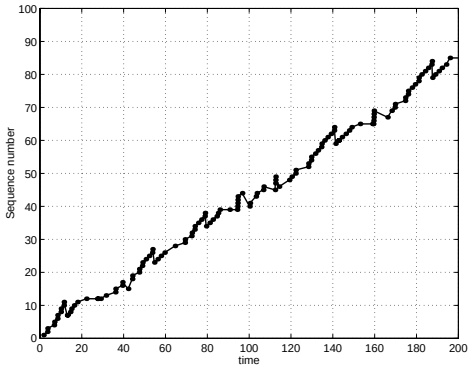

Figure 2.5. *TCP provokes packet losses and has no explicit feedback.*

The measurements were generated by a simulation model implemented in the MatLab applications Simulink and Stateflow. The complete model is illustrated and discussed in Appendix A. The simulations have been done in a one-way scenario to simplify the outcome, i.e. only one of the hosts had any data. This host always had data to send during the connections.

The network was modeled with three queues of different lengths and their waiting times were linear functions of the packetsize plus a Gaussian error. Since the packets only occupy one cell in the queue, no matter what their size are, the waiting time as a function of the size corresponds to larger packets occupying more space than small ones.

We see the way TCP probes the network in Figure 2.5, the figure shows the sequence numbers of the packets sent as a function of the sending time. The sending rate (slope) varies and does not seem to converge. E.g. at time 80 and for the sequence numbers between 30 and 40 we see a high sending rate followed by a retransmission (a 'dip'), i.e. a provoked packet loss. This is Problem 1 mentioned in Section 2.2.

Problem 2 regards the use of the bandwidth. In our simulation testbed we only used one link and only one TCP connection. Therefore, the available bandwidth is the same as the total bandwidth. The limiting factor is the length of the smallest queue. At each time instant, our system can carry and process at most as many packets as this queue can store simultaneously. This means that, in this case, the bandwidth use is the same as the occupancy of the smallest queue, the bottleneck queue. A long queue corresponds to large use of the bandwidth and vice versa. The distribution of the

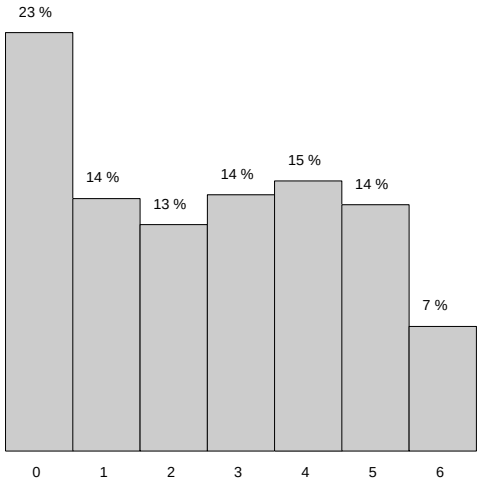

Figure 2.6. *The use of the bandwidth oscillates. In this special case the available bandwidth is the same as the length of the shortest queue. The histogram shows the distribution of the bottleneck queue length, during active connections.*

bottleneck queue length, during active connections, is shown in Figure 2.6. We see that the queue is empty for almost one forth of the time and there is no specific peak, which would be the case if the system reached equilibrium. If we consider a setup with constant incoming rate until a packet is lost and constant output rate, the outermost position of the queue will only be occupied half the time compared with the other positions. Thus, what we see in Figure 2.6 is that TCP fills the queue until a packet is lost and then stops sending until the queue is empty and a little longer.

Even when no packets are lost, TCP might experience problems. Figure 2.7 shows the sent packets during a transmission when no packets were lost. TCP retransmitted some packets anyway. We also see TCP's Time-Out limit, TOL, based on the round trip time estimate, and the experienced round trip time. Remember that TCP calculates the round trip time estimate using a low pass filter (see Section 2.1). This means that TCP will have problems when adjusting to fluctuations in the round trip time, due to e.g. varying network load.

The retransmissions shown Figure 2.7 come from false timeout indications. In several places, we see that the round trip time experienced by TCP are higher than the timeout limit. After a while we hope for that TCP will find the steady state and for these oscillations to stop. Though,

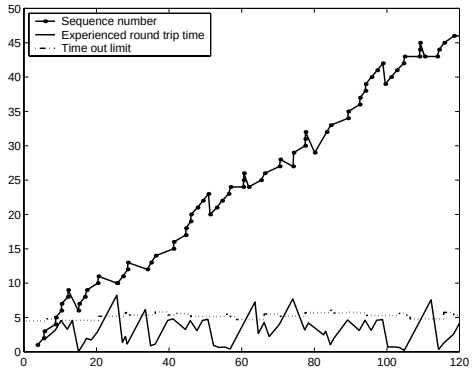

Figure 2.7. *All congestion notifications may not be due to a packet loss. Here is the sender side during a part of a transmission when no packets were lost. The retransmissions occur because of large variations in the experienced round trip time. The time out limit is shown for comparison.*

during short transmissions and when the load of the network varies, this non-adaptivity might become a big problem.

Another behavior that has been observed during the simulations is that a retransmission caused by three duplicate acks is most often followed by a timeout. This is one of the things that causes 13 % of the transmissions to be retransmissions even when only 5% of the packets are lost, see Problem 5. After a timeout TCP retransmits all the packets sent during the last round trip time.

2.4 Resent Research Concerning TCP

Internet has scaled remarkably, from four nodes in 1969 to almost 50 million hosts today. In resent years, there has been a lot of research concerning Internet and the unavoidable problems with such a fast growth. Research has been done on

- router modelling in e.g. [7, 10]
- traffic modelling, Internet traffic arrivals are modelled as being Poisson distributed, [3, 8]
- how to improve TCP,
- performance improvements without involving TCP. There are suggestions to:

- enhance the routing algorithms, [4]
- solve the problem with traffic control as a global optimization problem, [17]
- share information about the state of the network between local area of hosts, [24, 28]

In Appendix B you will find a summary of some of the articles on these different topics. In this report we will focus on the improvements of TCP.

Floyd [12, 13] has done extensive research on Explicit Congestion Notification, ECN. This means that in some way you explicitly tell the hosts that the network is congested, mainly by marking the packets. One way is to use the average queue size to determine whether or not to mark the packets.

Her work has been continued by Sisalem & Schulzrinne [30, 31] who proposed Binary Congestion Notification, BCN, which means that the way to tell the hosts is by setting one bit according to some rules. Their conclusion is that there might be of some interest to explore Explicit Rate Notification, ERN, instead [29]. With ERN you tell the host how much bandwidth it should be using to avoid congestion.

Mascolo et.al. [14, 21, 22] etc. consider a feedback solution for congestion control. This scheme has been investigated for high speed networks, ATM and (of course) for TCP. Mascolo [21] proposes a solution with a Smith predictor to improve TCP's performance. For the calculations he models the network using a delay, an integrator and a second delay, (see Section 3.2).

Johansson & Karlsson [18, 19] combines Explicit Rate and feedback control. Their algorithm improves the performance of TCP's retransmissions during slow start. The work is an extension to both Mascolo's and Floyd's work and we will only consider one of them in this report.

Since Mascolo's proposal is the suggestion that has been developed from a feedback control perspective, with the origin at the existing protocol, this is what we are going to focus on. We will describe his results in the following chapter, and also look at their impact on our simulation model.

Chapter 3

TCP with Feedback

First we will give a short description of the Smith predictor, then we will go through the steps in Mascolo's work and finally compare the results from simulations in our model with and without the proposed changes.

3.1 The Smith Predictor

The Smith predictor is mainly used on systems with large delays, since such systems are hard to control with ordinary methods. Otto Smith [32] proposed a solution that extracts the time delay out of the controller.

Consider a system with transfer function $G(s)e^{-sT}$. We want to control this system with a controller, $R(s)$, such that the resulting closed loop transfer function is $F(s)e^{-sT}$, see Figure 3.1, i.e. we want to find $R(s)$ such that

$$G_c(s) = \frac{R(s)G(s)e^{-sT}}{1 + R(s)G(s)e^{-sT}} = F(s)e^{-sT} \tag{3.1}$$

To do this, we first construct a controller, $\tilde{R}(s)$, such that the closed loop

$$\tilde{G_c}(s) = \frac{\tilde{R}(s)G(s)}{1 + \tilde{R}(s)G(s)} \tag{3.2}$$

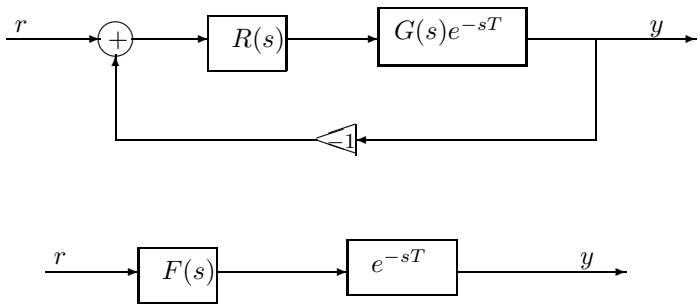

Figure 3.1. *Smith's predictor, $R(s)$, moves the delay out from the control loop*

has nice properties, see for example Glad & Ljung [16] on basic control theory. Now, we let $F(s) = \tilde{G}_c(s)$ and we can calculate $R(s)$ by using Equation (3.1).

$$F(s)e^{-sT} = \tilde{G}_c(s)e^{-sT} = \frac{\tilde{R}(s)G(s)}{1 + \tilde{R}(s)G(s)}e^{-sT} \tag{3.3}$$

and

$$G_c(s) = \frac{R(s)G(s)e^{-sT}}{1 + R(s)G(s)e^{-sT}} \tag{3.4}$$

Combining Equations 3.1, 3.3 and 3.4 we get

$$R(s) = \frac{\tilde{R}(s)}{1 + (1 - e^{-sT})\tilde{R}(s)G(s)}. \tag{3.5}$$

This controller gives the closed loop system, $G(s)$, the same dynamical behavior as $\tilde{G}_c(s)$, since the only difference is a time delay. The stability and robustness properties might change.

3.2 Flow Control with a Smith Predictor

Now let us see how this can be applied on TCP. This section is completely based on Mascolo's discussion in [21, sections 4 and 6].

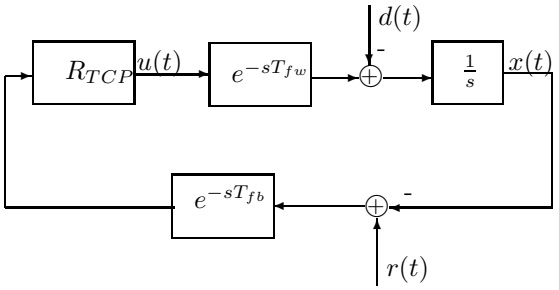

Figure 3.2. *Model of TCP controlled network flow*

3.2.1 The Network Model

First we need to have a model of the network. Here the network is going to be our system and TCP the controller. The bottleneck queue is the queue with the slowest service rate. Therefore the bottleneck queue of the system is the only queue greater than zero. In Figure 3.2 the model is described with the following notation:

- The input rate to the network, $u(t)$.
- An integrator which models the bottleneck queue.
- The output from the bottleneck queue, $x(t)$. This is the integrated value of the transmission rate, which corresponds to the length of the queue.
- The forward delay, T_{fw}, from the sender to the bottleneck queue.
- The feedback time delay, T_{fb}, from the bottleneck queue to the receiver and back to the sender.
- A disturbance, $d(t)$, which models the available bandwidth. Mascolo also assumes that it is not possible to measure $d(t)$.
- The controller transfer function, $R_{TCP}(s)$.
- Finally we have the reference signal, $r(t)$. This corresponds to the total size of the bottleneck queue available for this particular flow.

Thus, we have the connection between r and x:

$$x = \frac{1}{s}\left(e^{-sT_{fw}} R_{TCP}(s) e^{-sT_{fb}}(r - x) - d\right) \tag{3.6}$$

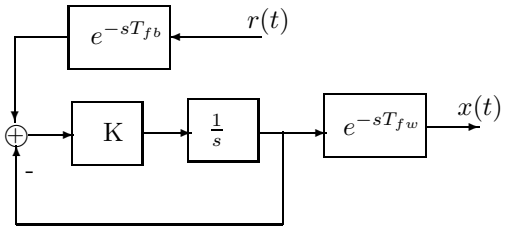

Figure 3.3. *The equivalent system, using a Smith predictor.*

and after some calculation we get the transfer function from r to x as:

$$x = \frac{\frac{1}{s}e^{-sT_{fw}}R_{TCP}(s)e^{-sT_{fb}}}{1+\frac{1}{s}e^{-sT_{fw}}R_{TCP}(s)e^{-sT_{fb}}}r = \frac{R_{TCP}(s)\frac{1}{s}e^{-sT_{rt}}}{1+R_{TCP}(s)\frac{1}{s}e^{-sT_{rt}}}r \tag{3.7}$$

where $T_{rt} = T_{fw} + T_{fb}$ represents the total round trip time.

If we compare this expression with the discussion in Section 3.1, and especially Equation (3.4) we see that $R_{TCP}(s)$ corresponds to $R(s)$ and $\frac{1}{s}$ corresponds to $G(s)$. Thus, we need to find a controller, $\tilde{R}(s)$, to nicely control the integrator. The simplest way is to let $\tilde{R}(s) = K$, where K is a constant. When using Equation (3.5) with $G(s) = \frac{1}{s}$ and $\tilde{R}(s) = K$ we get

$$R_{TCP}(s) = \frac{K}{1+(1-e^{-sT_{rt}})K\frac{1}{s}}. \tag{3.8}$$

This gives us the resulting closed loop system shown in Figure 3.3.

This is how TCP should perform if you follow Smith's proposal, and use proportional controlling for the system without the time delay. In the time domain this corresponds to the input rate control equation

$$u(t) = K(r(t-T_{fb}) - x(t-T_{fb}) - \int_{t-T_{rt}}^{t} u(\tau)d\tau). \tag{3.9}$$

This means that the input rate, $u(t)$, is proportional to the space left in the buffer, $r(t-T_{fb}) - x(t-T_{fb})$, decreased by the packets sent during the last round trip time, i.e. the packets that are sent but not yet acknowledged.

Mascolo has also carried out the mathematical analysis to show that this control law satisfies stability and utilization conditions, i.e. it ensures that the bottleneck queue is not overflowed and also that it is never empty.

3.2.2 TCP and a Smith Predictor

Since the control law of TCP is window based we need to transform Equation (3.9) to a window based equation. Mascolo does this by considering the sample time, T_s. We can then interpret the amount of data $u(t)T_s$ as a *Window*, W, of data that can be sent at time t. Equation (3.9) can be rewritten as follows

$$\mathsf{W} = u(t)T_s = T_s K \left(r(t - T_{fb}) - x(t - T_{fb}) - \int_{t-T_{rt}}^{t} u(\tau) d\tau \right). \tag{3.10}$$

The window W will represent an impulse of data, sent at time t. At every sample time, W data is sent. This is asumed to happen instantly.

In the previous discussion about TCP's congestion control, we defined the outstanding packets as the packets sent by the *sender* but not yet acknowledged. Therefore we have, with Mascolo's notation

$$\mathsf{PacketsOut} = \int_{t-T_{rt}}^{t} u(\tau) d\tau. \tag{3.11}$$

As Mascolo points out, this integral comes from the Smith predictor and it follows that TCP already implements a Smith predictor, though developed through heuristic arguments. The remaining quantity $r(t-T_{fb})-x(t-T_{fb})$ is the space left in the bottleneck queue, i.e. it is the *minimum free buffer space* over the connection, including the receiver's buffer. If we denote the free space remaining in the *i-th* buffer by, B_i, we can see that

$$r(t - T_{fb}) - x(t - T_{fb}) = \min\{B_i, \mathsf{AdvWnd}\}. \tag{3.12}$$

This is the *Generalized Advertised Window*, GAW. The minimum is taken over all the buffers encountered by the TCP connection on its path from the *sender* to the *receiver*. We can now rewrite Equation (3.9) as

$$\mathsf{W} = T_s K(\mathsf{GAW} - \mathsf{PacketsOut}) \tag{3.13}$$

and for $K = 1/T_s$

$$\mathsf{W} = (\mathsf{GAW} - \mathsf{PacketsOut}) \tag{3.14}$$

This means that when the *sender* receives a GAW in a packet, W packets are sent. Since we do not need the round trip time to compute the number of outstanding packets, the window control equation automatically takes into account variations in RTT. The GAW is stored where today's TCP

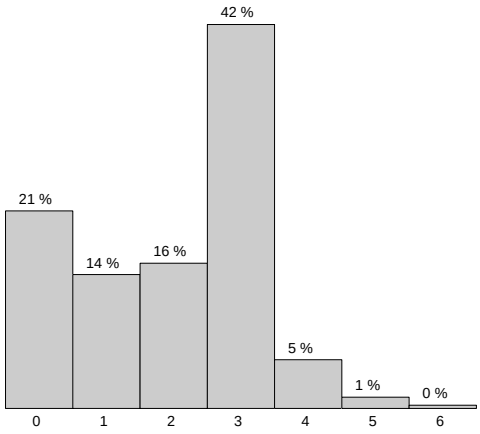

Figure 3.4. *The bottleneck queue length.*

stores the AdvWnd. Thus, combining the flow control equation (2.1) and Equation (3.14) we get

$$\mathsf{MaxSend} = \mathsf{W} = \min\{\mathsf{cwnd}, \mathsf{GAW}\} - \mathsf{PacketsOut} \tag{3.15}$$

To implement this in the today's TCP, we only need to make sure that the routers stamp the size of the space left in the queue if it is smaller than the existing value. Then TCP's flow control can run unchanged using Equation (3.15) instead of Equation (2.1). This version of the TCP will be referred to as *feedback TCP*.

3.3 Performance of a Feedback Solution for TCP

Mascolo's suggestion was easily added to the used model, and we will now look at the performance. The control law ensured that the bottleneck queue would never be overflowed, i.e. no packet losses. This is also confirmed by the simulations. This means that problem 1 is addressed.

One of the problems with TCP was the oscillating bandwidth use. Figure 3.4 shows the histogram over the bottleneck queue length for active connections with feedback. We see a peak and that the queue is almost never full. We also see that there is still a large amount of time when the queue is empty. Thus, compared with ordinary TCP (Figure 2.6 on page 13), we have improved the use of the bandwidth, but not optimized it.

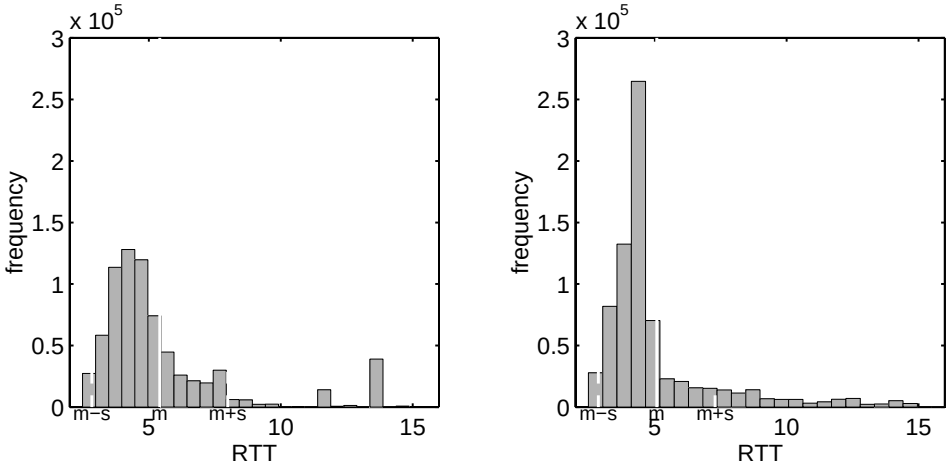

Figure 3.5. *The round trip time estimate for ordinary TCP (left) and feedback TCP.*

In theory the feedback control scheme should make sure that the bottleneck queue is never empty. We see that this is not the case. This is probably caused by the cwnd in Equation (3.15). Ordinary TCP had troubles with adapting to changes in the round trip time, and since the feedback TCP uses the same updating algorithm for the round trip time estimate this will also be the case here. Thus the false timeouts will keep the cwnd on a lower level than necessary until we find some kind of equilibrium.

Also, we have seen that the time delay of the TCP connection causes problems. This time delay is estimated by the hosts, during a connection, and stored as an internal parameter, the RTT estimate. The histograms over these estimates for the two different implementations are shown in Figure 3.5. The means and the standard deviations are indicated with the white vertical lines. Table 3.1 also shows the numerical values of the means and the variances for the two roundtrip time estimates. We see that the value of the estimate is slightly smaller and not as spread for the feedback TCP as it is for the ordinary TCP. Ordinary TCP also experiences congestion, which result in large delays, as we can see.

Figure 3.6 shows the sender side during a connection with the same setup as for the ordinary TCP in Figure 2.5. We see that after approximately 50 packets, the equilibrium is reached. Remember that this setup only considers one TCP connection and no shared bandwidth. This means that the available bandwidth for this connection is constant. In reality, when the

Table 3.1. *The means and the variances for the round trip time estimates of the two TCP implementations.*

For ordinary TCP:	mean, m_1,	5.4133
	variance, v_1,	6.6302
For feedback TCP:	mean, m_2,	5.0751
	variance, v_2,	4.9069

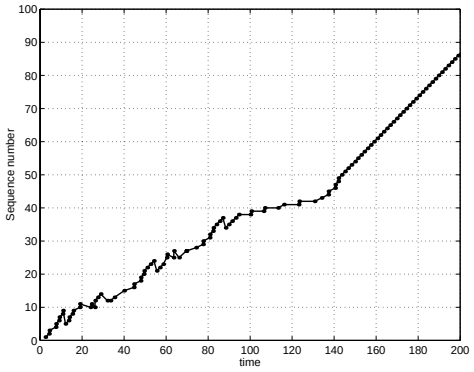

Figure 3.6. *The sender side of a feedback TCP connection.*

number of TCP connections sharing the same bottleneck queue varies over time, this adaptiveness is good.

We have seen that feedback TCP performs better than ordinary TCP. We also saw that the theory and practice did not always agree. E.g. the feedback scheme did not, in practice, ensure full utilization. Some questions still remain unanswered:

a. what impact would a discrete solution have,
b. what happens when the scheme is used in multiple connections,
c. what happens when both ordinary TCP and feedback TCP are used in the same network,
d. could the value of K, $\neq \frac{1}{T_s}$, be tuned to increase performance,
e. could another controller, $\tilde{R}(s) \neq K$, be used.

To answer these questions we need to

i. build a discrete mathematical model,
ii. extend the SIMULINK model,
iii. implement the solutions in a real Internet environment, and
iv. calculate the solutions when using different controllers.

Chapter 4

Conclusions

We have seen that Internet is a broad area of research, only on improving the control properties there are numerous directions and proposals. A significant amount of work has been done on improving the performance of TCP. There are two major directions that have made more progress than others: Proposals to change or extend the way to notify TCP about congestion and proposals to apply automatic control theory. Both require changes in the routers and some minor additions to the protocol. There is also a proposal that combines these two. The work with a new way of congestion notification has experienced some problems with fairness towards other control algorithms. Competing connections using another control algorithm might not get their appropriate bandwidth share. The work with a control theoretic background has come up with a reliable proposal.

Simulations of this latter proposal compared with the ordinary TCP show improvements, the major one being that we get no packet losses for a single connection. It also decreases the system delay and stabilizes the traffic load. We conclude that feedback control can be very useful in a communication system such as the Internet, despite the time delays.

When studying the Internet, we are dealing with a discrete-time system. Thus, it would be interesting to model this fact. The feedback solution is, as we have seen, based on a continuous-time model of the traffic flow. The feedback solution also uses a proportional controller and the performance

might very well be increased by tuning the value of the proportional constant or even using another controller. The simulation model could also be extended to show the behavior of multiple connections, since we do not know how ordinary and feedback TCP work together. In the existing TCP we have a good base with a lot of improvement aspects to consider.

Appendix A

Modeling TCP in Simulink and StateFlow

In this section we will describe the model that has been used for carrying out simulations. The model was made using SIMULINK and STATEFLOW in MATLAB 6.0. Documentation for these programs can be found in [1]. In STATEFLOW modeling is based upon states and events. Different events and conditions define when transitions between states can occur. To make it easier to understand in the following model description, we will now use Figure A.1 to explain some things about STATEFLOW. The corresponding STATEFLOW-expressions to each description are described in the parenthesis after each explanation.

- There are two states, *Main/* and *Noconnect/*.
- *Noconnect/* is the default start state (the arrow with a loose starting point).
- A transition from *Noconnect/* to *Main/* occurs on the event *UpStart* and when this happens the output data *ackno* is set to the input data *seqno* +1 (/ackno=seqno+1;).
- When we are in *Main/* the *ackno* is updated according to a MATLAB function "ack1" that takes the arguments ackno and seqno (ackno=ml('ack1(%g,%g)',ackno,seqno)).

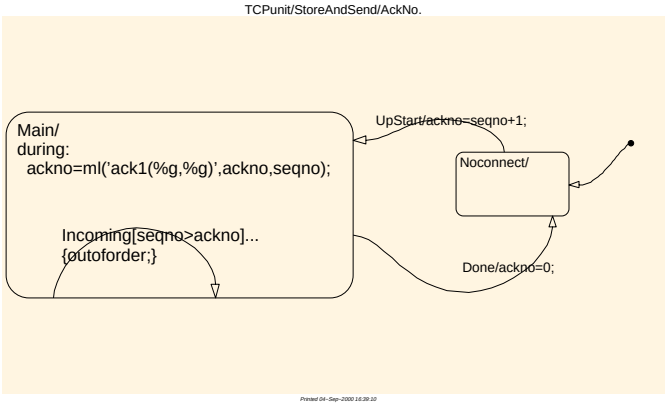

Figure A.1. StateFlow *block for reference.*

- On the event *Incoming* and if the *seqno* is larger than the *ackno* the inner transition will occur and the event *outoforder* will be broadcasted (Incoming[seqno>ackno]{outoforder;}).

- On the event *Done* we will go from state *Main/* to state *Noconnect/* and *ackno* will be set to zero (Done/ackno= 0;).

- Later on we will also see that states can have dashed borders. This means that these states can be active at the same time, "AND configuration".

We will start on the top layer of the model and work our way down. In Figure A.2 the structure of the model is shown as a tree. Square boxes represent Simulink blocks and boxes with rounded corners represent StateFlow blocks. If the model is carefully examined there will appear to be unnecessary steps, especially in the StateFlow blocks. Many of these have been added during simulations due to the performance of StateFlow.

A.1 The TCP Unit

As shown in Figure A.2, the TCP unit consists of three main blocks. One for the internal parameters, one for contact with other TCP blocks and one block to take care of the discrete events. The three blocks are connected to each other as shown in Figure A.3.

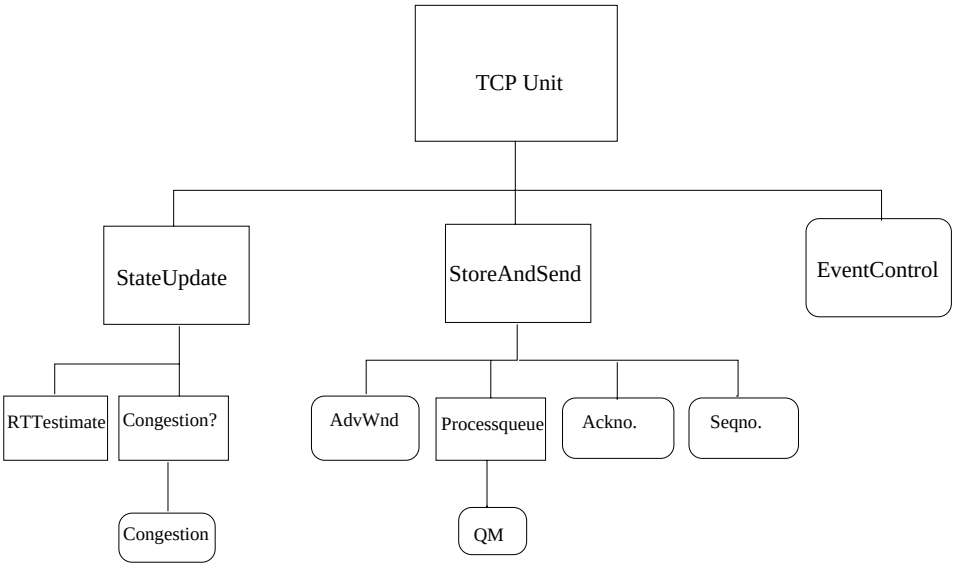

Figure A.2. *The TCP model's structure. Sharpcornered boxes are* SIMULINK *blocks and rounded boxes are* STATEFLOW *blocks. They each handle one or more of the parts of TCP. (QM=Queue manager)*

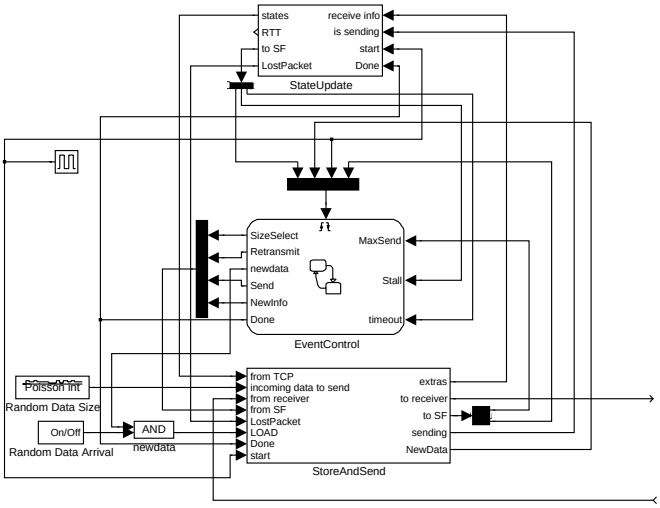

Figure A.3. *The TCP unit. The top layer of the model.*

Table A.1. *The different outputs from StoreAndSend*

Name	Content	Recipient
extras	AdvWnd, ack no. and size from the incoming packet	*StateUpdate*
to receiver	Outgoing packet	TCP unit
to SF	The data *MaxSend* and the events *UpStart*, *Exit*, *LastData*, *Incoming* and *OutOfOrder*	*EventControl*
sending	The packet most recently sent	*StateUpdate*
NewData	The event *NewData*	*EventControl*

A.1.1 StoreAndSend

The *StoreAndSend* block is shown in detail in Figure A.4. This is where the communication takes place. *StoreAndSend* takes the incoming messages and sends the different information to different blocks of the TCP unit. It also receives a lot of information from the other blocks. The different output and input signals are listed in Tables A.1 and A.2, together with a short description. In the following description a *process* is the data that is going to be transmitted to the receiver, e.g. a web site or an ftp-file. Here, such a process is represented only by its size.

On the top level *StoreAndSend* keeps track of the size of the packets, with help of the data *SizeSelect* from *EventControl*. *StoreAndSend* also checks whether the whole process is sent, by knowing the number of the first packet and the size of the whole process and the size of each packet.

AckNo.

The block *AckNo.* is only keeping track of the incoming packets sequence numbers and computes the appropriate acknowledgment number. The entire block is shown in Figure A.5.

SeqNo.

The block *SeqNo.* updates the sequence number every time a new packet is sent, see Figure A.6.

Table A.2. *The different inputs to StoreAndSend*

Name	Content	Sender
from TCP	The internal parameters, e.g. cwnd, ssthresh and MaxSend	*StateUpdate*
incoming data to send	simulates the size of a new process	Poisson process
from receiver	Incoming packet	Other TCP unit
from SF	the data *Retransmit* and *SizeSelect* and the events *Send* and *NewInfo*	*EventControl*
LostPacket	The lost packet that is going to be retransmitted	*StateUpdate*
LOAD	Event that controls when the TCP unit is ready for a new process	*EventControl* & On/Off process
Done	Event when the transmission is complete	*EventControl*
start	the clock pulse	square wave

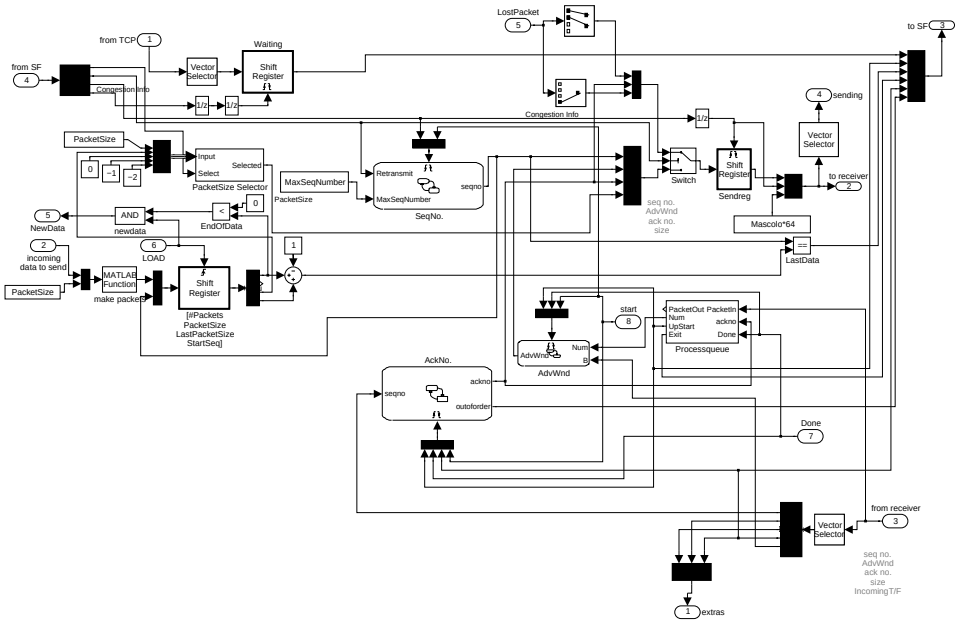

Figure A.4. *StoreAndSend takes care of the communication with other hosts.*

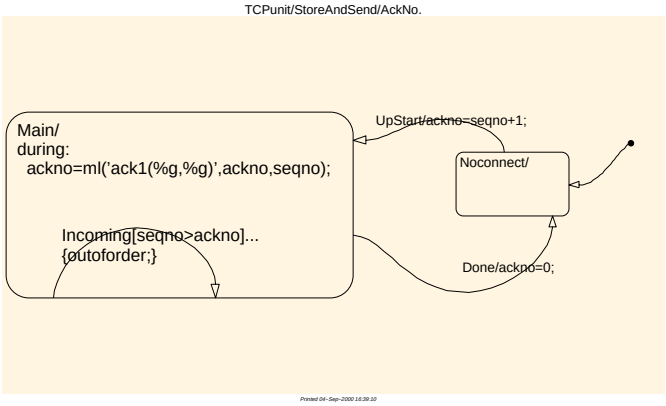

Figure A.5. *StoreAndSend/AckNo.*

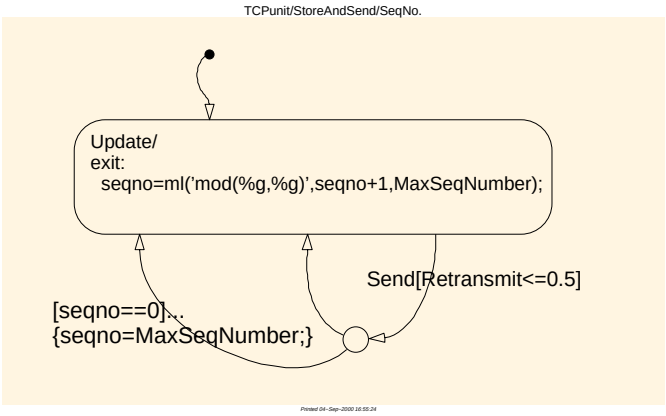

Figure A.6. *StoreAndSend/SeqNo.*

AdvWnd

Depending on whether we are using the model proposed by Mascolo or not, the block *AdvWnd* computes the Advertised Window accordingly, see Figure A.7.

Processqueue

The *Processqueue* (Figure A.8) simulates the time for the host to process the incoming data. A sequence of data can not be processed unless all

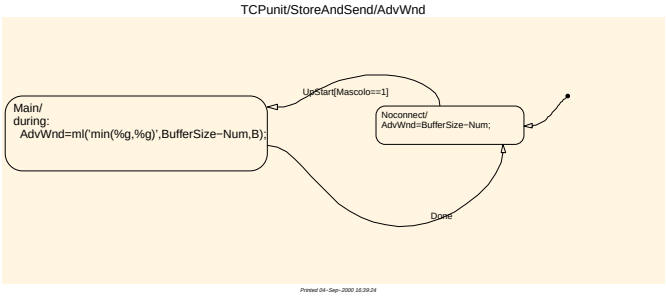

Figure A.7. *StoreAndSend/AdvWnd*

sequences before have arrived, i.e. the packets have to come in the correct order. The packets that arrived in the wrong order are stored in the queue *saveforlater* until all the sequences before have arrived, i.e. until the host's acknowledgment number is larger than the sequence number. The 'in-order' data packets and their corresponding process times are located in the two other queues.

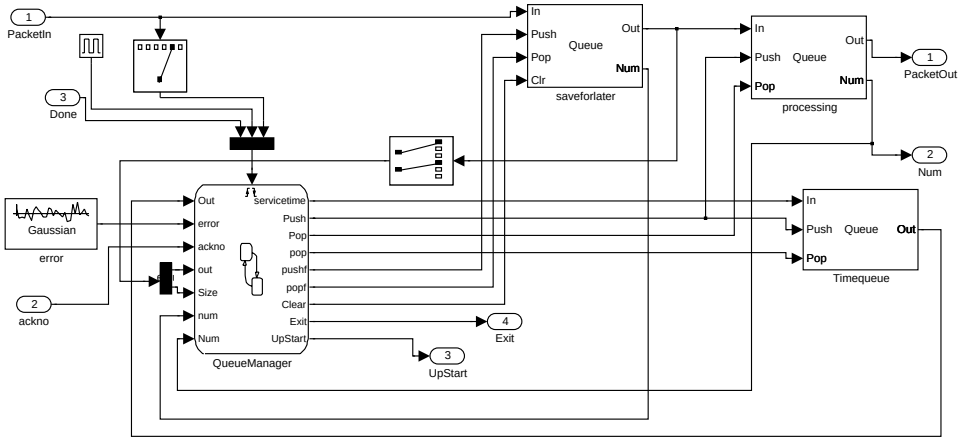

Figure A.8. *StoreAndSend/Processqueue*

QM The block *QM* (short for queue manager) calculates the process time for each packet and 'Pops' and 'Pushes' on appropriate times for the three different queues. The waiting time is a linear function of the size plus a

Gaussian error. This models the fact that the hosts will use more time for large packets than for small ones.

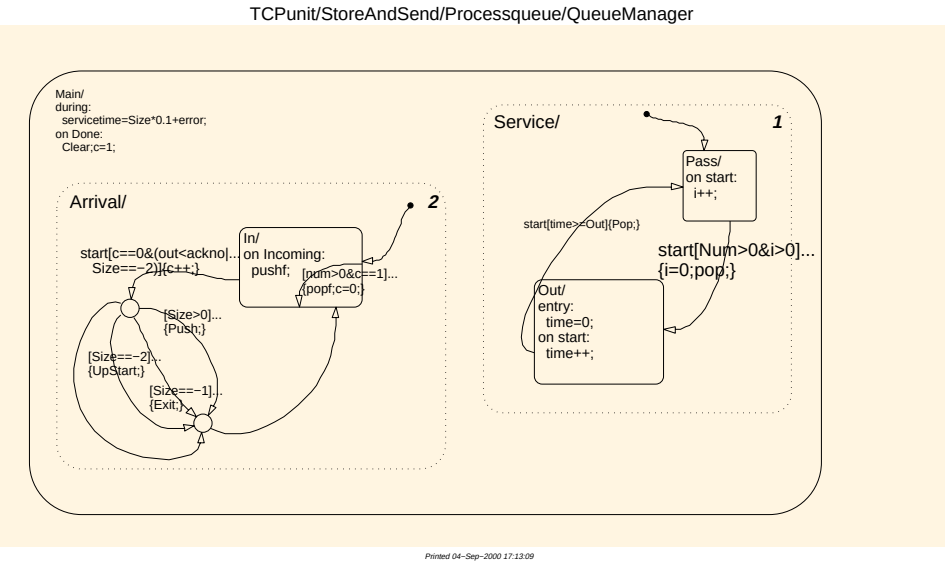

Figure A.9. *StoreAndSend/Processqueue/QM*

A.1.2 StateUpdate

The block *StateUpdate* keeps all the inside information. It also communicates with the other blocks in the TCP unit and the different outputs and inputs to the block are listed in Tables A.3 and A.4.

The main task for *StateUpdate* is to use the incoming information to detect congestion. For this it uses two subblocks. On the top level, see Figure A.10, *StateUpdate* stores some information and sends other information to the other blocks in the unit.

Congestion?

In the *Congestion?* block (Figure A.11) the sent packets are stored until a corresponding ack is received. This is to make it possible to retransmit the original packet when congestion occurs. The send time is also stored to decide whether we have a Timeout or not. When using SIMULINK to implement this, the drawback is that you cannot pick an arbitrary packet

Table A.3. *The different outputs from StateUpdate*

Name	Content	Recipient
extras	AdvWnd, ack no. and size from the incoming packet	StateUpdate
states	cwnd, ssthresh, MaxSend and the number of duplicate acks plus some extra parameters for internal control	*StoreAndSend*
RTT	The calculated round trip time	workspace
to SF	The events *LostPacket* and *ackE* and the data *Stall* and *timeout*	*EventControl*
LostPacket	The lost packet that is going to be retransmitted	*StoreAndSend*

Table A.4. *The different inputs to StateUpdate*

Name	Content	Sender
receive info	Information from the last received packet, ack no., AdvWnd, and the event *Incoming*	*StoreAndSend*
is sending	The last sent packet and the event *Sending*	*StoreAndSend*
start	the clock pulse	square wave
Done	Event when the transmission is complete	*EventControl*

from the queue to retransmit, but you have to simply send the next in line. In most cases this will be the correct one, but there are special cases that will cause some problems.

Congestion The block *Congestion* stores the internal parameters, cwnd, ssthresh, MaxSend and PacketsOut and also updates them through the simulation. In Figure A.12 we see that different parts of the block are responsible for different actions. Every time a message arrives all the parameters are updated. Different updating rules apply in different situations, which corresponds to the different possible paths in the substate *Arrival/*.

To notify the rest of the unit about congestion the parameter *lostpacket* is used. *lostpacket* also enables the output from the *Queue* in the superblock *Congestion?*, from the start of the congestion until the lost packet has been

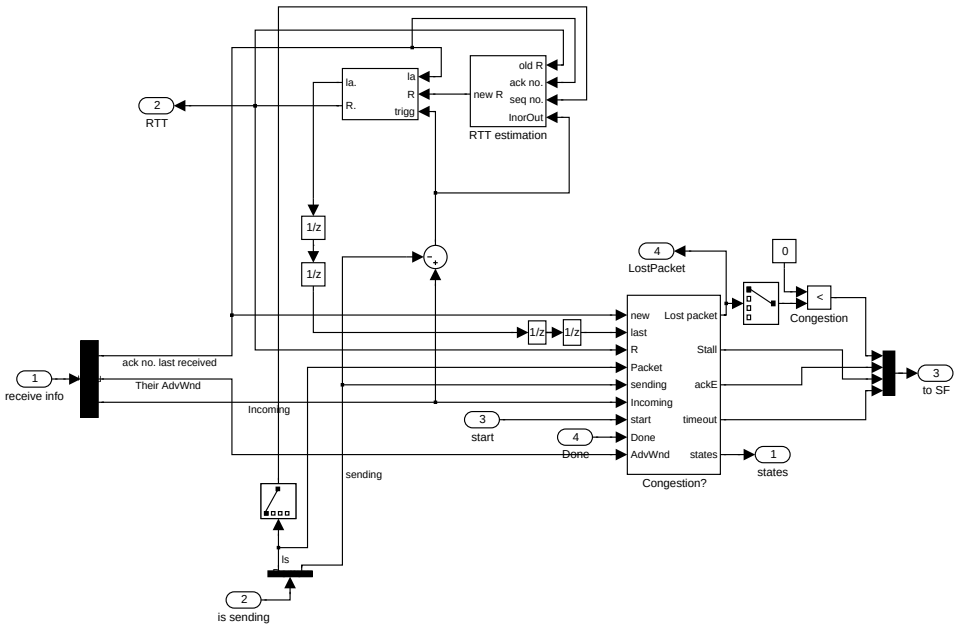

Figure A.10. *StateUpdate handles the internal parameters.*

retransmitted. During Timeout the TCP unit continues to retransmit until all the packets have been retransmitted or until a new packet arrives, i.e. one round trip time. This is ensured by the substate *Around/* in *Pop/*.

The parameter *Stall* is only used to make sure that every part of the unit is done with everything before the next packet is sent.

RTT estimation

In *RTT estimation* TCP's calculation of the round trip time is carried out. A sequence number is saved until the corresponding ack arrives and then a new value of the RTT is calculated. Figure A.13 shows the SIMULINK implementation of this.

A.1.3 EventControl

The block *EventControl* decides when things happen and if they are allowed to happen, e.g. when a sending takes place. In Figure A.14 we see that different things happen in different phases of the transmission. The

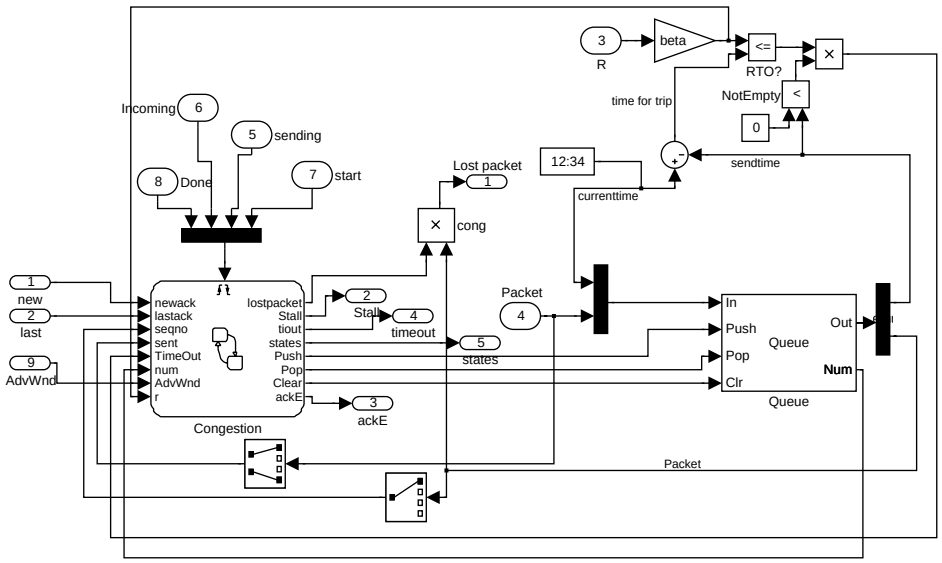

Figure A.11. *StateUpdate/Congestion?*

transmission can either be down, initializing, running or exiting. The parameter *MaxSend* and the event *Congestion* are used to stop the transmission or start it from somewhere else. The little circle with an H in it remembers the previous configuration of the state, i.e. when exiting the state *Congestion* we will automatically continue from where we were in *Main/* when the event *Congestion* took place. If we have data to send, a packet is sent on every clock pulse and if we have no data a packet is sent as a response of incoming ones.

A.2 The Network Model

Since this model was built to show TCP's behavior, the model of the network is very simple: one link with three "routers" in each direction. As mentioned in section 2.3 the waiting times in the routers are functions of the packet size. This is to simulate that the packets occupy one cell per bit in reality.

A.2.1 Networkqueue & MascoloQueue

To implement the feedback control scheme proposed by Mascolo as discussed in section 3.2 only a few changes were needed in the "network". In

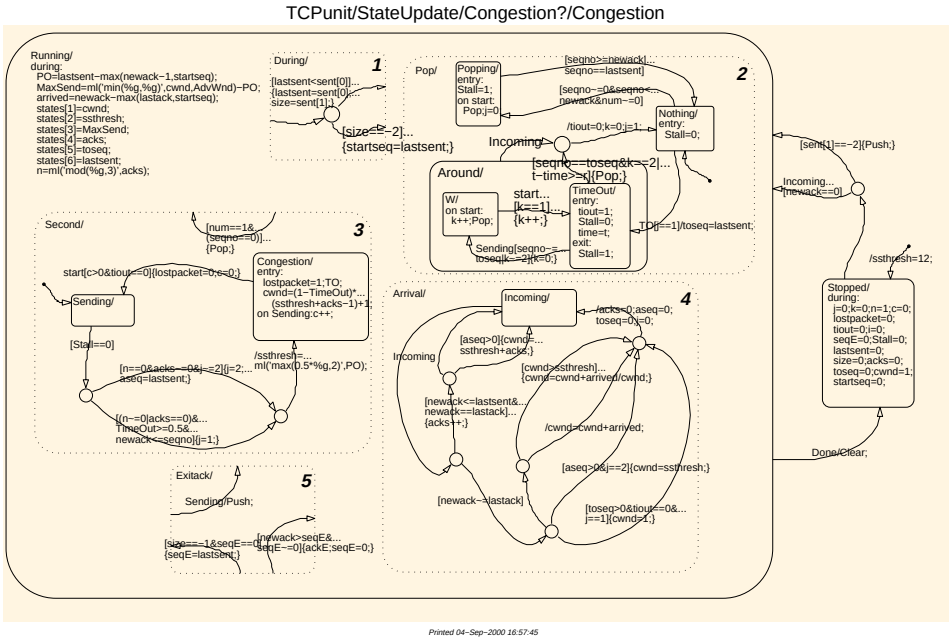

Figure A.12. *StateUpdate/Congestion?/Congestion*

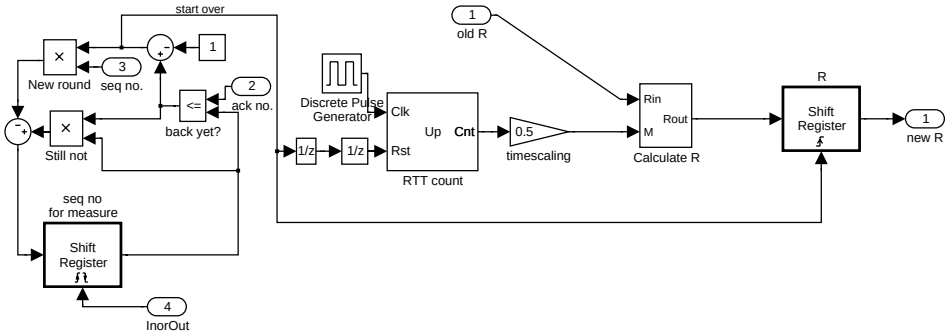

Figure A.13. *StateUpdate/RTT estimation*

Figures A.15 and A.16, we see the similarities between the two versions. The incoming packets and their corresponding waiting times are stored and in *MascoloQueue* the packets are 'stamped' if their stored value is greater than the space left in this queue.

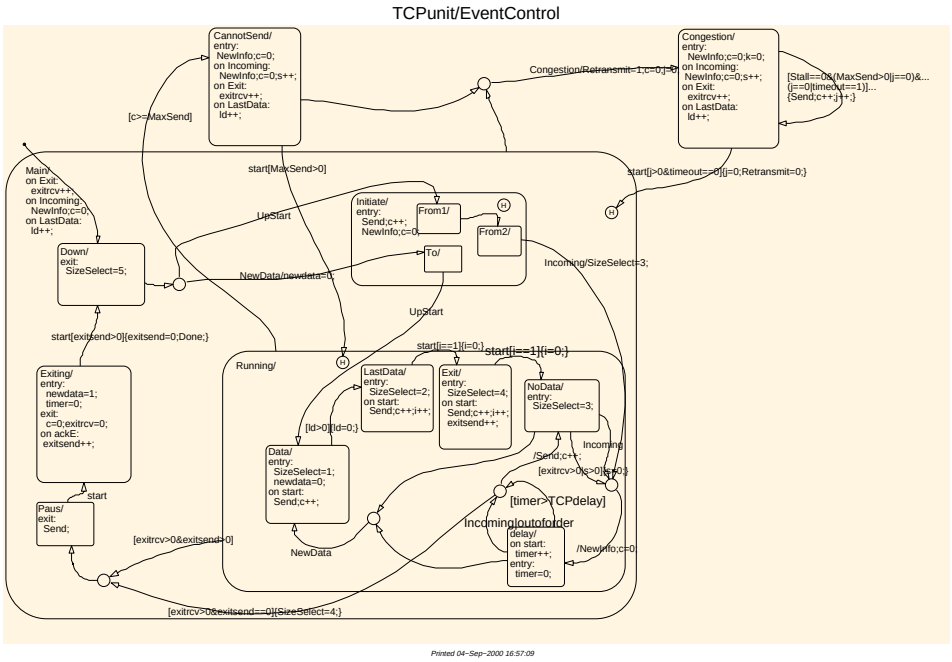

Figure A.14. *EventControl controls the discrete events.*

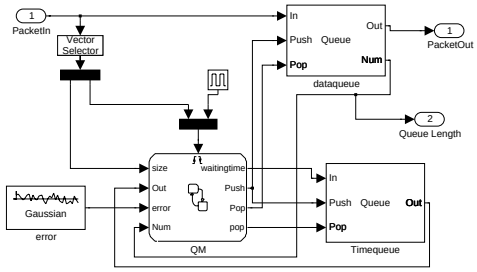

Figure A.15. *Networkqueue used when modeling TCP as it is.*

QueueManager

The *QueueManager* (Figure A.17) calculates the waiting time and performs 'Pops' and 'Pushes' to the two queues in the superblock, at appropriate times. The block *QueueManager* in the *MascoloQueue* works in the same way.

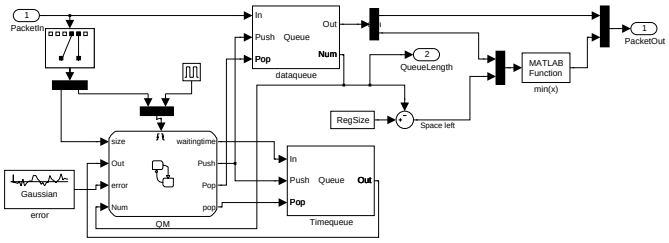

Figure A.16. *MascoloQueue used when modeling TCP with a Smith predictor.*

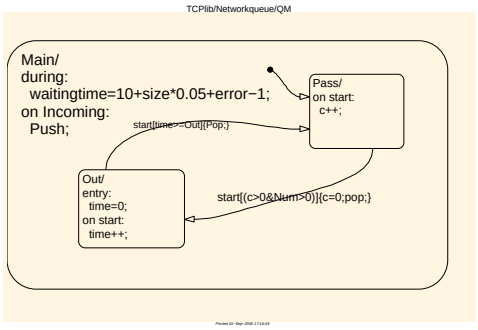

Figure A.17. *Networkqueue/QueueManager*

Appendix B

A Summary of the Research in the Area

In this chapter some articles from different areas of research are briefly summarized. When possible, the control parameters are listed and the addressed problem from Section 2.2 is indicated. Also, we try to summarize the work in each research area. Table B.1 lists commonly used terms in this chapter, with a short explanation.

B.1 Traffic Modeling

A Markovian Approach for Modeling Packet Traffic with Long-Range Dependence, Andersen and Nielsen [3]:

The paper models packet traffic with superpositions of two-state MMPP's. (MMPP = Markov Modulated Poisson Process). An MMPP is a counting process with dependent counters (two-state means two counters). Each counter is Poisson distributed. The result is itself a MMPP, a special case of a Markov Arrival Process, MAP.

The M/G/1 Queue with Heavy-Tailed Service Time Distribution, Boxma and Cohen [8]:

Long range dependence in a traffic process is modeled with a fluid queue fed by one or more on/off sources. The on-period has nonexponential heavy-tail behavior: $Pr[A > t] \overset{t\to\infty}{\sim} h_\varsigma t^{-\varsigma}$. Explicit expressions are derived for the

Table B.1. *Explanations to common terms.*

Available Bit Rate service	allows non-real time data traffic to use bandwidth that is assigned to real time traffic but temporarily left unused.
end-to-end	we do not care about the interior of the network, only the end hosts
fairness	the share of the bandwidth should not depend on the used protocol
heterogenous	different packets have different requirements concerning reliability and timing
on/off source	is either one or zero with some distributions for the up or down events
Reno & Tahoe	different versions of the TCP, Reno is the latest of the two
traffic class	e.g. video streams, ftp files, web pages.
transport-level	The level of the network where the control is done, usually only at the end hosts.

M/G/1-queue. (The arrival rate is Markovian, the service rate is general and it is a single queue.)

Summary of section. The prevailing traffic model is some kind of on/off process with different mean arrival rate and holding times for different traffic classes. Mostly the arrivals are considered Poisson distributed. See also Section B.2.

B.2 Router Modeling

Virtual Partitioning for Robust Resource Sharing: Computational Techniques for Heterogeneous Traffic, Borst and Mitra [7]:

Virtual partitioning is a scheme for sharing a resource (e.g. a link or a buffer) among several traffic classes. First, each class is given a nominal capacity. The scheme will then give higher priority to underloaded classes. It is efficient, fair and robust. Expressions for blocking probabilities are derived. It models traffic as Markov On/Off.

- controllers: blocking, acceptance region, required capacity, reservation mechanisms, link occupancy, (priority).
- measures: blocking rate, link occupancy.

A Framework for Optimizing the Cost and Performance of Next-Generation IP Routers, Chan et al. [10]:

The paper develops an analytical framework to model and analyze the impact of technological factors on cost performance in distributed routers. The main goal is to divide the decisions between an ensemble of forwarding engines, instead of one centralized engine per router. It discusses two router models, distributed and parallel router architecture and it models traffic as Poisson.

B.3 Flow and Congestion Control

B.3.1 TCP Improvements

Changes in the TCP The following articles discuss different changes that can be made in the TCP to improve the performance on congestion and flow control.

TCP behavior of a busy Internet server: analysis and improvements, Balakrishnan et al. [5]:

Designed new application-independent transport-level mechanisms. TCP-INT (Integrated Congestion Control/Loss Recovery) improves the performance of multiple TCP connections. Each host has a single congestion window for all its parallel TCP connections with a server. Measurements on a busy Web server show that 6% of the packets are lost, 13% of the transmissions are retransmissions and 85% of the packets are sent during slow start.

- controllers: combined cwnd.
- measures: time of last decrease wnd.
- addresses problem: 2.

Random Early Detection gateways for Congestion Avoidance, Floyd and Jacobson [13]:

Random Early Detection, RED, is used in routers to notify the network about congestion. It uses the average queue size (avg) to determine whether or not to mark the packets. If the avg is between some values, then the packets are marked with a certain probability, if it is higher they are all marked. The problem is to determine the optimum of these values, as they are heavily dependent on network characteristics.

- addresses problem: 1 (2).

Characteristics of an Explicit Rate ABR Algorithm, Johansson and Karlsson [18]:

This paper evaluates a previously proposed control algorithm. The algorithm combines both the information about the buffer occupancy ([21]) and the input rate ([29]) to control an Available Bit Rate, ABR, service.

Interaction Between TCP Flow Control and ABR Rate Control, Johansson et al. [19]:

This study considers the interaction between the TCP flow control and the ABR Explicit Rate control. A comparison between the algorithm from [18] and the ordinary TCP algorithm is made. The results indicate that TCP's behavior after timeouts can be improved using the explicit rate algorithm.

Smith's predictor for congestion control in TCP Internet protocol, Mascolo [21]:

The paper proposes that Smith's predictor is used to design the congestion control algorithm. It reveals that the flow and congestion control algorithm in today's TCP already has a Smith's predictor. It uses Generalized Advertised Window to obtain feedback.

- controls: input rate of TCP connection.
- measures: bottleneck queue length.
- addresses problem: 1,6 (2).

Binary congestion notification in TCP, Sisalem and Schulzrinne [30]:

The paper continues as proposed by Floyd [12]. It uses Binary Congestion Notification, BCN, to enhance the congestion control mechanism in TCP at the gateways. Switches inform the sources about their congestion state by setting a congestion bit in the data packets. It also discusses that Explicit Rate Indication might be a better way to enhance TCP. The issue of fairness is not investigated thoroughly enough.

- controllers: transmission wnd.
- measures: congestion bit, acks.
- limitations: RTT, efficiency dependent on switch algorithms.
- addresses problem: 1,(2).

Congestion control in TCP: performance of binary congestion notification enhanced TCP compared to Reno and Tahoe TCP, Sisalem and Schulzrinne [31]:

This paper discusses performance of BCN-TCP from [30] compared with Reno and Tahoe. BCN is a good integration base for TCP and available

bit rate service, ABR, but it suffers from some fairness problems. The performance on heterogenous traffic is not known. Explicit rate notation might be worth checking out.

Changes around the TCP The following articles present different improvements in the network without changing the TCP.

An end-system architecture for unified congestion management, Rahul et al. [24]:

The Congestion Manager, CM, is described. The CM maintains congestion and path related information and allows flows to learn from each other. The paper separates the loss recovery function from congestion management. The main components are a rate-control scheme similar to TCP's, implicit feedback from e.g. acknowledgment packets and a flexible flow scheduler. CM ensures that all traffic adheres to basic Internet congestion control principles. CM also considers the interaction between active flows, unlike HPF.

The CM is application independent and only a mean to share information. Every application can use the CM-functions to find out statistics of the network.

- addresses problem: 2, 3 .

(*RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet*, Rejaie et al. [25]):

The goal is to make realtime streams good network citizens. The idea is to separate congestion control from error control. The paper presents the design of Rate Adaption Protocol, RAP. RAP adopts to the AIMD algorithm, RAP performs loss-based rate control. RAP is TCP-friendly when TCP uses mostly AIMD, i.e. is mostly in the congestion avoidance phase. It is a core component of an end-to-end network.

Summary of section. Two major directions can be found. The papers on Smith's predictor and GAW [14, 21] discuss the matter of feedback and also show results compared to other algorithms. Also the papers about explicit congestion notification [13, 30, 31] show nice results.

B.3.2 Solutions without TCP

MPLS and traffic engineering in IP networks, Awduche [4]:

Discusses the application of Multiprotocol Label Switching, MPLS, to traffic engineering in IP networks.
Network = demand system (traffic) + constraint system (interconnected network elements) + response system (network protocols and processes).
Separate different stages for the traffic engineering process: 1. control policy formulation, 2. network state observation, 3. traffic characterization + network state analysis, 4. network performance optimization.

MPLS allows routing control capabilities to be introduced in IP networks. MPLS simplifies the network design compared to an overlay model. The route is determined at the origination node. The model consists of four basic elements: Path management, traffic assignment, network state information variation and network management.

- controls: labelswitched paths (LSP).
- measures: detection of congestion.

Load Balancing and Control for Distributed World Wide Web Servers, Castro et al. [9]:

This paper describes a load balancing and control algorithm for multicomputer Web servers. The algorithm ontrols the request rate to web server engines and makes full processing capacity of each engine available. It requires no prior knowledge of relative speeds of the engines, nor the work required for each request. (Eddie Open Source Project)

- controls: fraction of accepted sessions to each server, fraction of rejected sessions.
- measures: processor load/queue length.

An adaptive transport protocol for multimedia communication, Dwyer et al. [11]:

Describes new transport protocol, HPF, for supporting heterogenous packet flows. Separates policies (controlled by the application) and mechanisms (controlled by the transport layer). Support heterogeneity in frame/packet level and provide effective congestion control in transport layer. HPF uses the fraction of received packets to control congestion. (TCP uses last ack).

- controls: transmission wnd, "priority".
- measures: fraction of received packets.
- limitations: urgent≠reliable, no delay bound.
- addresses problems:4,6.

A class of end-to-end congestion control algorithms for the Internet, Golestani and Bhattacharyya [17]:

End-to-end control of user traffic as a global optimization problem. Comes up with a class of congestion control algorithms. Develops theoretical framework to address several topics of increasing importance. Their Minimum Cost Flow Control (MCFC) algorithm is compared with TCP Reno.

HPF: a transport protocol for supporting heterogeneous packet flows in the Internet, Li et al. [20]:

Describes HPF from [11] with more test results.

Flow control and bandwidth management in next generation Internets, Pazos et al. [23]:

Presents a link layer flow control using: label swapping in the routers, Class Based Queuing link sharing and ABR Virtual Path Connections.

- measures: Explicit Rate(ER).
- limitations: doesn't prevent congestion in destination routers.
- addresses problems: 2.

The case for informed transport protocols, Savage et al. [28]:

Propose an approach in which congestion information is shared within a local area of hosts (e.g. an organization). This avoids TCP's slow start.

- controls: transmission rate(overall).
- measures: RTT measured by other hosts, acks across connections.
- internal parameters: dropped packet locality.
- addresses problems: 3,(2).

A Queue Growth Rate Based Flow Control Algorithm, Yao and Doray [34]:

Propose a feedback flow control algorithm for ABR flows in ATM networks. The algorithm uses explicit rate to control the network. See also [23].

- controls: allowed cell transfer rate (ACR).
- measures: RTT, queue growth rate, link input rate.
- addresses problems: 1.

Summary of section. Here we find three different explicit control algorithms. HPF, [11, 20], which uses the fraction of received packets to decide further actions. With ER, [23, 34], you use the input rate and queue growth rate to calculate parameters and decide the allowed cell transfer rate. From the EddieOpenSource project comes an algorithm for distributed Web servers, [9], that uses load or queue length to update fraction of accepted sessions to each server and fraction of rejected sessions.

B.4 Overview - and - Summary - Articles

On TCP performance in a heterogeneous network: a survey, Barakat et al. [6]:

The paper summarizes problems that face TCP in the Internet of today, independent of network type. Main problems are burstiness and coupling between congestion detection and error control.

Highlights of Signal Processing for Communications, Giannakis [15]:

This collection of articles celebrates the highlights from 50 years of signal processing technology as it has been applied to communication systems. Presents articles on: Filter Banks for Signal Representation and Source Coding. Fractal Geometry and Nonlinear Dynamics in Communication. Channel Estimation, Equalization and Synchronization. Voiceband Modems: A Signal Processing Success Story. Antenna Arrays for Wireless Networks. Co-Channel Signal Separation and Adaptive Beamforming. Issues in Military Communications. Time-Frequency for Interference Excision in Spread-Spectrum Communications. Multicarrier Communications. Multi-User Communications/Multi-User Detection. Signal Processing for Communication Networks.

Understanding next generation Internet, Rutkowski [26]:

Next Generation Internet, NGI, is divided into seven sectors; 1. Basic Technologies; 2. Devices; 3. Networked Platforms; 4. Infrastructure; 5. Infostructure; 6. Metastructure; 7. Network Management.

Detour: informed Internet routing and transport, Savage et al. [27]:

This paper describes inefficiencies in routing and transport protocols in the modern Internet and attempts to quantify these effects. It discusses a prototype called Detour, a virtual Internet, in which routers tunnel packets over the commodity Internet instead of using designated links. Detour is a virtual network testbed to explore the costs and benefits of informed routing and transport mechanisms.

Bibliography

[1] http://www.mathworks.com.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control, April 1999. RFC 2581.

[3] A.T. Andersen and B.F. Nielsen. A Markovian Approach for Modeling Packet Traffic with Long-Range Dependence. *IEEE Journal on Selected Areas in Communications*, 16(5), Jun 1998.

[4] D.O. Awduche. MPLS and traffic engineering in IP networks. *IEEE Communications Magazine*, 37(12):42–47, Dec 1999.

[5] H. Balakrishnan, V.N Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP behavior of a busy Internet server: analysis and improvements. In *IEEE INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, volume 1, pages 256–262, 1998.

[6] C. Barakat, E. Altman, and W. Dabbous. On TCP performance in a heterogeneous network: a survey. *IEEE Communications Magazin*, 38 (1):40–46, Jan 2000.

[7] S.C. Borst and D. Mitra. Virtual Partitioning for Robust Resource

Sharing: Computational Techniques for Heterogeneous Traffic. *IEEE Journal on Selected Areas in Communications*, 16(5), Jun 1998.

[8] O.J. Boxma and J.W. Cohen. The M/G/1 Queue with Heavy-Tailed Service Time Distribution. *IEEE Journal on Selected Areas in Communications*, 16(5), Jun 1998.

[9] M. Castro, M. Dwyer, and M. Rumsewicz. Load Balancing and Control for Distributed World Wide Web Servers. In *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, volume 2, pages 1614–1619, August 1999.

[10] H.C.B. Chan, H.M. Alnuweiri, and V.C.M. Leung. A Framework for Optimizing the Cost and Performance of Next-Generation IP Routers. *IEEE Journal on Selected Areas in Communications*, 17(6), Jun 1999.

[11] D. Dwyer, S. Ha, J.-R Li, and V. Bharghavan. An adaptive transport protocol for multimedia communication. In *Multimedia Computing and Systems, 1998. Proceedings. IEEE International Conference on*, pages 23–32, Jul 1998.

[12] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, Oct 1994. [This issue of CCR incorrectly has "1995" on the cover instead of "1994".].

[13] S. Floyd and V. Jacobson. Random Early Detection gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[14] M. Gerla, R. Lo Cigno, S. Mascolo, and W. Weng. Generalized Window Advertising for TCP Congestion Control. Technical Report 990012, UCLA Computer Science Departement, Feb. 1999.

[15] G.B. Giannakis(Editor). Highlights of Signal Processing for Communications. *IEEE Signal Processing Magazine*, 16(2):14–50, Mar 1999.

[16] T. Glad and L. Ljung. *Reglerteknik, Grundläggande teori*. Studentlitteratur, Lund, second edition edition, 1989.

[17] S.J. Golestani and S. Bhattacharyya. A class of end-to-end congestion control algorithms for the Internet. In *Sixth International Conference on Network Protocols, 1998. Proceedings.*, pages 137–150, 1998.

[18] P. Johansson and J.M. Karlsson. Characteristics of an Explicit Rate ABR Algorithm. In *ITC Specialists Seminar on Control in Communications*, Sep. 1996.

[19] P. Johansson, E. Wedlund, and J.M. Karlsson. Interaction between TCP flow control and ABR rate control. In *IEEE ATM Workshop 1997. Proceedings*, pages 455–464, May 1997.

[20] J.-R. Li, S. Ha, and V. Bharghavan. HPF: a transport protocol for supporting heterogeneous packet flows in the Internet. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, volume 2, pages 543–550, 1999.

[21] S. Mascolo. Smith's predictor for congestion control in TCP Internet protocol. In *Proceedings of the 1999 American Control Conference, 1999.*, volume 6, pages 4441–4445, 1999.

[22] S. Mascolo, D. Cavendish, and M. Gerla. ATM rate-based congestion control using a Smith Predictor. *Performance Evaluation*, 31:51–65, Nov 1997.

[23] C.M. Pazos, M. Gerla, and G. Rigolio. Flow control and bandwidth management in next generation Internets. In *1998 1st IEEE International Conference on ATM, 1998. ICATM-98.*, pages 123–132, 1998.

[24] H.S. Rahul, H. Balakrishnan, and S. Seshan. An end-system architecture for unified congestion management. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 52–57, 1999.

[25] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, volume 3, pages 1337–1345, 1999.

[26] A.M. Rutkowski. Understanding next generation Internet. *IEEE Communications Magazine*, 37(9):99–102, Sep 1999.

[27] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, and E. Hoffman. Detour: informed Internet routing and transport. *IEEE Micro*, 19(1):50–59, Jan-Feb 1999.

[28] S. Savage, N. Cardwell, and T. Anderson. The case for informed transport protocols. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, 1999.*, pages 58–63, 1999.

[29] D. Sisalem and H. Schulzrinne. End-to-End Rate Control in ABR. In *First Workshop on ATM Traffic Management*, pages 281–287, Dec. 1995.

[30] D. Sisalem and H. Schulzrinne. Binary congestion notification in TCP. In *Communications, 1996. ICC '96, Conference Record, Converging Technologies for Tomorrow's Applications. 1996 IEEE International Conference on*, volume 2, pages 772–776, Jun 1996.

[31] D. Sisalem and H. Schulzrinne. Congestion control in TCP: performance of binary congestion notification enhanced TCP compared to Reno and Tahoe TCP. In *1996 International Conference on Network Protocols, 1996. Proceedings.*, pages 268–275, 1996.

[32] O.J.M. Smith. A controller to overcome dead time. *ISA Journal*, 6(2): 28–33, 1959.

[33] W.R. Stevens. *TCP/IP illustrated vol 1:The protocols.* Addison Wesley, 1994.

[34] Z. Yao and B. Doray. A Queue Growth Rate Based Flow Control Algorithm. In *Communications, Computers and Signal Processing, 1999 IEEE Pacific Rim Conference on*, pages 357 – 360, Aug 1999.