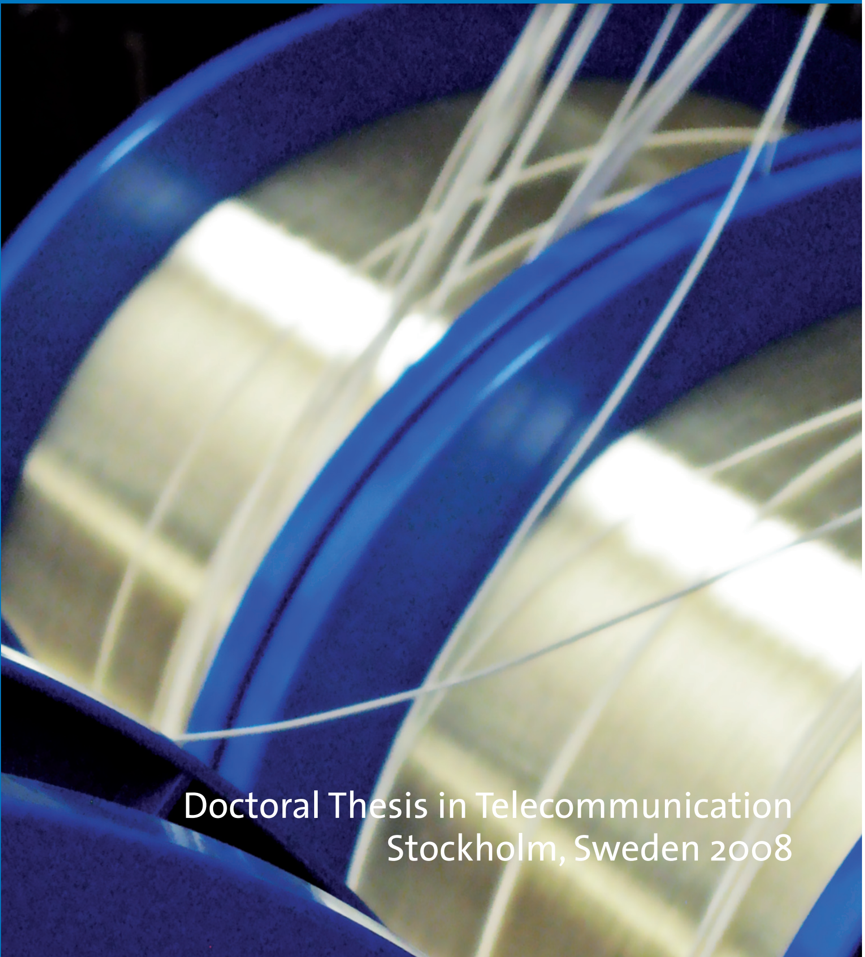
Dynamic modeling of Internet congestion control

KRISTER JACOBSSON



Doctoral Thesis in Telecommunication
Stockholm, Sweden 2008

KTH Electrical Engineering

KTH Electrical Engineering

Dynamic modeling of Internet congestion control

KRISTER JACOBSSON

Doctoral Thesis
Stockholm, Sweden 2008

TRITA-EE 2008:020
ISSN 1653-5146
ISBN 978-91-7178-945-7

KTH School of Electrical Engineering
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungliga Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i telekommunikation fredagen den 9 maj 2008 klockan 10.15 i sal F2, Kungliga Tekniska högskolan, Lindstedtsvägen 26, Stockholm.

Tryck: Universitetsservice US AB

Peace

Abstract

The Transmission Control Protocol (TCP) has successfully governed the Internet congestion control for two decades. It is by now, however, widely recognized that TCP has started to reach its limits and that new congestion control protocols are needed in the near future. This has spurred an intensive research effort searching for new congestion control designs that meet the demands of a future Internet scaled up in size, capacity and heterogeneity. In this thesis we derive network fluid flow models suitable for analysis and synthesis of window based congestion control protocols such as TCP.

In window based congestion control the transmission rate of a sender is regulated by: (1) the adjustment of the so called window, which is an upper bound on the number of packets that are allowed to be sent before receiving an acknowledgment packet (ACK) from the receiver side, and (2) the rate of the returning ACKs. From a dynamical perspective, this constitutes a cascaded control structure with an outer and an inner loop.

The first contribution of this thesis is a novel dynamical characterization and an analysis of the inner loop, generic to all window based schemes and formed by the interaction between the, so called, ACK-clocking mechanism and the network. The model is based on a fundamental integral equation relating the instantaneous flow rate and the window dynamics. It is verified in simulations and testbed experiments that the model accurately predicts dynamical behavior in terms of system stability, previously unknown oscillatory behavior and even fast phenomenon such as traffic burstiness patterns present in the system. It is demonstrated that this model is more accurate than many of the existing models in the literature.

In the second contribution we consider the outer loop and present a detailed fluid model of a generic window based congestion control protocol using queuing delay as congestion notification. The model accounts for the relations between the actual packets in flight and the window size, the window control, the estimator dynamics as well as sampling effects that may be present in an end-to-end congestion control algorithm. The framework facilitates modeling of a quite large class of protocols.

The third contribution is a closed loop analysis of the recently proposed congestion control protocol FAST TCP. This contribution also serves as a demonstration of the developed modeling framework. It is shown and verified in experiments that the delay configuration is critical to the stability of the system. A conclusion from the analysis is that the gain of the ACK-clocking mechanism dramatically increases with the delay heterogeneity for the case of an equal resource allocation policy. Since this strongly affects the stability properties of the system, this is alarming for all window based congestion control protocols striving towards proportional fairness. While these results are interesting as such, perhaps the most important contribution is the developed stability analysis technique.

Acknowledgments

Although I am the single author of this work I am not the only contributor. In fact there are many people who in different ways have been instrumental in the completion of this thesis. I would like to express my gratitude to you.

Let me start with bringing my advisor Professor Håkan Hjalmarsson into the limelight. Sincerely thank you for your guidance during these years. It has been very instructive having you as a role model. The accuracy in the way you do research is impressive, not to mention the speed. I believe the mix of competent supervision and academic freedom you have provided has been very successful. I hope I get the opportunity to continue to collaborate with you in the future.

This also applies to my co-advisor, Professor Karl Henrik Johansson, who also deserves a special acknowledgment. Your enthusiasm and your "sky-is-the-limit" attitude is very encouraging. By just watching you from the side-line, I have learned the importance of collaboration and networking in research. I would also like to thank you and Håkan for the support and interest you have shown for my life beyond the PhD. Such things makes hard decisions easier.

To Professor Bo Wahlberg I am probably in debt with my life. Without you joining me in the hunt for decent lunches, I most likely would have starved to death by now. I would like to thank you for these years working for you. Your cheerful attitude to life and research pass on to your group. It has been a pleasure!

I would furthermore like to express my gratitude to Professor Steven Low for giving me the opportunity to visit and collaborate with his excellent research group at California Institute of Technology a couple of times during the last years. Many thanks also goes to his (former) team members Kevin Tang and Lachlan Andrew. It has been great working with and learning from you all. You made my stays at Caltech so much more fruitful and enjoyable.

All my colleagues at the Automatic Control group at KTH, I would like to thank as well. My math/linux/emacs/telecommunication guru, Niels Möller, definitely deserves an extra ACK. I am very grateful for your patience answering my endless stream of stupid questions. Also, thank you Jonas Mårtensson for leading the way since 1997 when we both started at KTH, it has been safe to surf on the wave a few months behind you. Märta Barenthin, thank you for reading parts of my manuscript. Karin Karlsson Eklund, thank you for running the place.

Many thanks to Mårran "Viktor" Kjellberg for doing some magic with the cover picture.

Finally, at last but not least, I would like to thank my beloved family who is the foundation of my life!

Contents

Chapter 1

Introduction

THE Internet, the worldwide "network of networks", has revolutionized the way we communicate. It has in only forty years or so grown exponentially from non-existing to ubiquitous, connecting more than 1.1 billion users today. In this communication era we have learned to "web browse", "e-mail", "chat", "file share", etc., on a daily basis. We have also discovered that the Internet can be used as an economically efficient communication infrastructure for pre-existing technologies such as, e.g, telephony and television, digital systems are now converging. The Internet is nowadays fully integrated with the society in the developed world, socially as well as economically, and it is hard to estimate its value and our dependence on it. Imagine the consequences if it crashed?

In 1986 the Internet suffered from a series of, so called, congestion collapses. Events where throughputs for applications were close to zero, even though network resources were fully utilized. The response of the Internet research community was to introduce congestion control—mechanisms where users' sending rates are adjusted to match the level of congestion in the network. These algorithms were developed in an iterative design process based on heuristics, small-scale simulations and experimentation. In the perspective of the violent evolutional process the Internet has been subject to since 1986, the achieved performance of these systems must be considered as extremely successful.

There are, however, indications that this evolutionary design path is reaching its limits and that new designs are needed in the near future. New wireless links, e.g., put new demands on the congestion control schemes, which current loss-based protocols do not meet. Furthermore, today's deployed algorithms suffer from difficulties in providing quality of service guarantees in terms of resource allocation and delay, as well as that the ubiquitous additive-increase-multiplicative-decrease (AIMD) algorithm that governs the congestion control today does not seem to scale well to high capacity networks. This is widely recognized and there is an intensive research effort ongoing, trying to find new congestion control designs suitable for a future Internet scaled up in size, capacity as well as heterogeneity.

1.1 Window based congestion control

In a packet switched network, such as the Internet, packets of data are sharing links with other traffic. Non-exclusive access to circuits is normative, however to the price of no user end-to-end capacity guarantees. Data packets are rather delivered on a "best-effort" basis. Briefly, when an endpoint sender transmits a file to a receiver in a packet switched network, the data is divided into chunks and packetized together with the destination address, the packets are then released into the network which is doing its best to deliver them to their intended destination.

A "best-effort" network does not provide any end-to-end packet delivery reliability. In window based congestion control, this is instead assured through feedback. The receiver at the destination acknowledges successfully received packets by sending an *acknowledgment packet* (ACK) to the sender. At an ACK arrival the sender decides what information (packet) that is to be (re-)sent and when.

Window based control algorithms regulate their rates by dynamically adjusting a *window*, which is an upper bound on the number of packets they are allowed to send before receiving an ACK. The time it takes from a packet is sent until it is acknowledged is called the *round trip time* (RTT). Since a window w amount of packets is sent during a RTT period of time τ, the *average sending rate* x during such an interval is given by the window size divided by the RTT, $x = w/\tau$. To be able to accommodate short term fluctuations in traffic load, network routers operate buffers. When the network gets congested buffers build up implying that packets may be subject to queuing delay on their route to their destination. Since the RTT τ consists of the total delay a packet is subject to when traveling through the network, this means that the sending rate x is not solely set by the sender by adjusting the window w, it is also dependent on the state of the network through the RTT τ.

A schematic picture of the control structure for window based congestion control is given in Figure 1.1. The endpoint protocol is at this level of detail represented by the three blocks: transmission control, window control, and congestion estimator. We observe that the feedback mechanism can be divided into two separated loops. In the outer loop the congestion estimator tries to estimate the level of congestion in the network. This estimate is used by the window control to adapt the window size to an appropriate level. The dynamics of the inner loop are given by the so called *ACK-clocking*. The transmission of new packets is controlled or "clocked" by the stream of received ACKs by the transmission control. A new packet is transmitted for each received ACK, thereby keeping the number of outstanding packets, i.e., the window, according to the specification of the window control.

The ACK-clocking mechanism is generic to all window based congestion control protocols, and subsequently the dynamics of the inner loop in Figure 1.1 are as well. The basic window based control problem thus consists of designing the dynamics of the outer loop in Figure 1.1, i.e., the window control and the congestion estimator, such that the overall network behavior is desirable. This design problem has received ample attention in the literature, while the impact and importance of

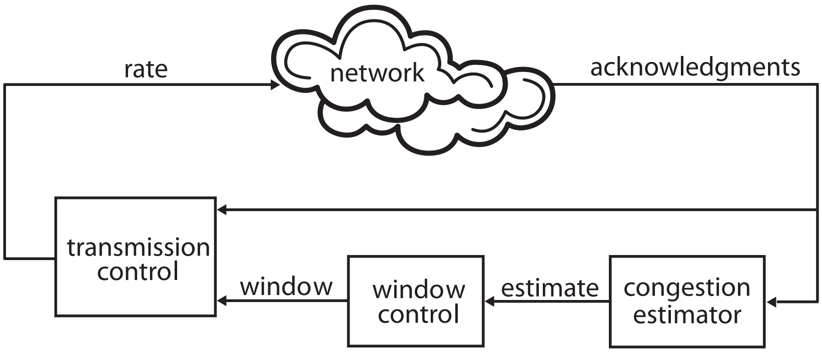


Figure 1.1: A schematic picture of window based congestion control.

the ACK-clocking dynamics on this problem largely has remained unrecognized.

1.2 Motivation

The *Transmission Control Protocol* (TCP) is the predominant transport protocol of the Internet today carrying more than 80 % of the total traffic volume (Fomenkov *et al.*, 2004). TCP is window based and to avoid network overload, which ultimately could cause congestion collapse, one of its primary roles is to adapt the sending rate based on the amount of congestion in the network. While TCP with great success has served as congestion controller on the Internet for years, it has started to reach its limits.

1.2.1 The need for speed

Let us investigate in a little bit more detail how TCP (Jacobson, 1988; Floyd *et al.*, 2004), scale with high capacity networks.

Consider a standard TCP flow operating at an average speed of a window w packets per RTT in a steady-state environment. It is well known today that for standard TCP, the average steady-state packet drop rate q of a flow approximately is

$$q = \frac{3}{2w^2}$$

packets per RTT, see, e.g., (Lakshman and Madhow, 1997; Mathis *et al.*, 1997). Let us assume a scenario with 1500 byte packets, quite standard on the Internet, and a 100 ms RTT, which roughly corresponds to the round trip propagation delay of a transatlantic connection. Achieving 10 Gbit/s of steady-state throughput would require an average speed of

$$w = \frac{10 \cdot 10^9}{1,500 \cdot 8} \cdot 0.100 = 8,3333$$

packets per RTT, and subsequently the average packet drop rate needed for full link utilization is

$$q = \frac{3}{2 \cdot 8,3333^2} \approx 2 \cdot 10^{-10}.$$

This corresponds to at most one congestion event about every $5,000,000,000$ packets, which in time is equivalent to at most one congestion event every 5/3 hours. An average packet drop rate of at most $2 \cdot 10^{-10}$ would correspond to a bit error rate of at most $2 \cdot 10^{-14}$—this is an unrealistic requirement for current networks (Floyd, 2003).

Evidently we can expect the performance of TCP to drop in future high bandwidth networks. The example furthermore highlights TCP's deficiency in wireless environments where packet drop (loss) rate typically is much higher than in a wired network. It is well-known that TCP, designed for a wired Internet, degrades over wireless links. Consequently, in pace with increased link capacities and as the Internet becomes more heterogeneous in terms of network technologies, the need for a replacement to TCP becomes more urgent.

1.2.2 The need for accurate models

Mathematical modeling is an useful tool to better understand the mechanisms behind a successful (or unsuccessful) congestion control design. However, the complexity of the Internet at packet level is tremendous, the key to tractable models is to abstract away irrelevant "details". Network fluid flow models, where packet level information is discarded and traffic flows are assumed to be smooth in space and time, are by now generally accepted as a viable route to analysis and synthesis of complex network communication systems. Following seminal work by Kelly *et al.* (1998), fluid flow modeling has been used in numerous studies on window based congestion control dynamics, e.g., in (Hollot *et al.*, 2001a; Low *et al.*, 2002b; Wang *et al.*, 2005; Tan *et al.*, 2006b; Peet and Lall, 2007; Möller, 2008). The validity of results concerning dynamical properties, however, rely heavily on the accuracy of the models.

The dynamics of the inner loop

Let us investigate the performance of some known fluid flow models of the dynamics of the ACK-clocking mechanism (the inner loop in Figure 1.1), often referred to as "the link" in window based congestion control. Consider the case of two window based flows with different RTTs sending over a single congested link (this is what we will call a *bottleneck*). The link buffer size, or queue, serves as the state of the

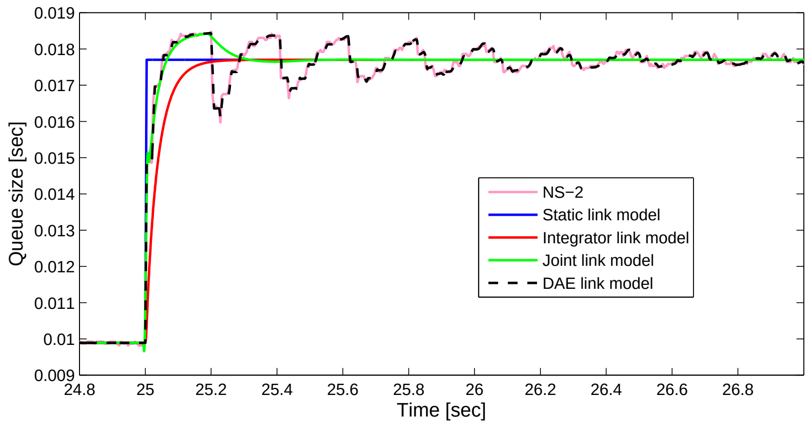


Figure 1.2: ACK-clocking dynamics for two flows sending over a single bottleneck link. Step response simulation. Basic configuration: Link capacity 150 Mbit/s. Packet size 1040 byte. Flow 1 propagation delay 10 ms. Flow 2 propagation delay 190 ms. Flow 1 window 210 packets. Flow 2 window 1500 packets. After convergence, at 25 seconds the first flows window is increased step-wise from 210 to 300 packets.

system. The system input, the window sizes, are constant initially (i.e., the outer control loop in Figure 1.1 is disabled). At 25 s the system is disturbed from the current equilibrium by a positive step-wise change in one of the window sizes. The plot in Figure 1.2 shows the response of the queue size for some previous models in comparison with the "true" queue size from a packet level simulation using NS-2 (ns2), and a model derived in this thesis. The "Integrator link model" appears in, e.g., (Hollot *et al.*, 2001a; Low *et al.*, 2002a; Baccelli and Hong, 2002; Altman *et al.*, 2004), the "Static link model" in, e.g., (Wang *et al.*, 2005; Wei *et al.*, 2006), and the "Joint link model" is taken from (Möller, 2008). We observe that they all fail to capture the significant oscillation in the queue. This is in contrast to the "DAE link model", that will be derived in Chapter 3, which shows almost perfect agreement with the packet level simulation, even at time scales finer than the RTTs (sub-RTT time scales).

The consequences for the outer loop

The analysis of the dynamic properties of a window based system based upon any of the previous models of the inner loop may yield qualitatively different results than those from the more accurate model. Especially for those versions of TCP which respond to short time scale dynamics of queue sizes (Brakmo *et al.*, 1994; Wei *et al.*, 2006; Tan *et al.*, 2006a). In particular, the oscillatory response shown in Figure 1.2 can cause such an algorithm to make rapid changes in its window size, resulting in increased jitter and reduced utilization. This will be confirmed for the special case of FAST TCP (Wei *et al.*, 2006) in Chapter 4.

Note that not all algorithms which respond to delay do so at these fast time scales. Algorithms such as TCP Africa (King *et al.*, 2005) and TCP Illinois (Liu *et al.*, 2006) operate on the time scale between loss events. However, even loss-based algorithms are indirectly affected by sub-RTT queuing phenomena. In particular, the degree of loss synchronization, which has a significant impact on the fairness of loss-based congestion control, is highly dependent on sub-RTT burstiness of flows. The more accurate ACK-clocking model developed in this thesis may help to predict such effects.

1.2.3 Discussion

We have pointed out that the microscopic behavior of the ACK-clocking mechanism, the fundament of all window based congestion control, may have a significant impact on the macroscopic performance of TCP. This is also established in the work by Wei (2007), who furthermore concludes that existing fluid flow models of TCP congestion control fail to capture important packet level phenomena, and thus predictions made based on such models may not be very accurate (as illustrated in Figure 1.2). Encouraged by this observation we will in this work derive fluid flow models of window based congestion control, with a more solid foundation in the originating packet level system than available before. In particular we will focus on protocols using delay as congestion notification, but many of the ideas do also apply to other types of algorithms. The model of the ACK-clocking, e.g., is valid for all window based schemes.

The modeling framework used here will be deterministic, and thus, e.g., packet loss effects are not in the scope of this thesis. While we agree that assuming no loss does not reflect the reality on the current Internet, we argue that it is of crucial importance to understand the basic mechanisms to be able to properly characterize a complex phenomena, such as packet loss due to buffer overflow, in the future. Considering, delay based congestion control the deterministic assumption is mild.

The price to pay for more accurate models is increased model complexity, and the use of fluid flow models is often motivated by their tractability. Detailed models, however, are, e.g., suitable for simulation. In addition, a precise model can be simplified to match the accuracy needed for the intended application, and is thus valuable. We believe these claims are substantiated in Chapters 4 and 5.

1.3 The thesis at a glance and contributions

The main contributions of the thesis appear in Chapters 3, 4 and 5 covering congestion control modeling, analysis and validation respectively. In more detail, the outline of the thesis is as follows.

Chapter 2: Background

In this chapter we provide an introduction to the Internet and the main principles it relies on. We outline the primary mechanisms that governs the congestion control and we discuss the control objectives. In particular the Internet congestion control working horse, the Transmission Control Protocol, is reviewed. An introduction to recent developments in congestion control modeling is given.

Chapter 3: Congestion control modeling

In this chapter we derive a fluid flow model of a network congestion control system. The emphasis is on the ACK-clocking mechanism and on window based congestion control protocols using queuing delay as network congestion notification.

A departure from traditional fluid flow congestion control models is that we go into sub-RTT detail and distinguish between the actual number of packets in flight, the flight size, and the desired number of packets in flight, the window size.

The main contribution of the chapter is a detailed model of the ACK-clocking mechanism, generic to all window based schemes. The model makes use of a fundamental integral equation which relates the instantaneous arrival rate at a link with the endpoint senders' flight sizes. This is the key in the modeling. Also non-window based flows are incorporated in the model as cross traffic. We also provide a description of the relation between the flight size and the window size.

To close the loop a novel fluid model of the dynamics of a general time based protocol is formulated. The model accounts for the relation between the flight size and the window size, window control, estimator dynamics as well as sampling effects that may be present in an endpoint congestion control algorithm. The use of the model is demonstrated by modeling FAST TCP.

Chapter 4: Analysis

This chapter is devoted to analysis of the models derived in the previous chapter.

First, we analyze the properties of the ACK-clocking mechanism. For a general network we show that it has a unique equilibrium. For the single bottleneck link case we are able to prove that the queue size is linearly stable around the unique equilibrium. We also illustrate that in the case of rational ratios between flows' RTTs, the system is not stable in the senders' rates. The individual rates actually may oscillate while their sum, and thus the queue, remains constant. The model predictions are verified with packet level validation experiments. This highlights how the ACK-clocking model captures microscopic phenomenon such as traffic burstiness. A discussion on some different strategies for how to reduce the complexity of the ACK-clocking model is also included.

Second, we investigate the quantitative difference between protocol window size and flight size. We see that for the typical case when the window size is updated once per RTT, the difference is in the order of a RTT.

Finally, we derive a linear model of multiple FAST TCP flows sending over a single bottleneck link. The linear model is used as departure point for finding more tractable simplified models. We use a model, valid for moderately heterogeneously distributed RTTs, to show that the system is stable for default parameter values. To do so we use a proof technique less conservative than previous state-of-the art in this type of work. However, since the model accuracy degenerates as the RTT heterogeneity increases we seek an alternative model to analyze such scenarios. After deriving such a model, we show that it is not possible to choose protocol parameters of FAST TCP such that system is stable in all networks. In particular it destabilizes when flows' RTTs are wide apart. The theoretical results are also confirmed with NS-2 simulations and testbed experiments.

Chapter 5: Experimental results and validation

In this chapter we validate the models derived in Chapter 3.

The chapter opens with a discussion on how to generate experimental data in closed loop for a window based congestion control system.

A quite extensive validation of the ACK-clocking model derived in Chapter 3 is presented next. This is done using experimental data from packet level simulations as well as from a physical testbed. The model is reported to be very accurate.

The dynamics from the window size to the flight size is investigated in simulation experiments. We observe a good fit between our model predictions and the experiments. For relative small changes in the window size the model error in the queue size when not accounting for the dynamics seems negligible, it is furthermore attenuated as the system converges to equilibrium.

We end the chapter with validating the FAST TCP model which the linear analysis in Chapter 4 is based on. The model fit is reasonable, in particular for high bandwidth scenarios which FAST TCP is primarily designed for.

Chapter 6: Conclusions and future work

The results of the thesis is summarized in this final chapter. We conclude the thesis by suggesting possible future extensions to the work presented.

1.4 Publications

Many of the results appearing in this thesis have been presented elsewhere. The material covered in Chapter 3, 4 and 5 is presented in part in the papers:

Krister Jacobsson, Lachlan L. H. Andrew, Ao Tang, Steven H. Low and Håkan Hjalmarsson. An improved link model for window flow control and its application to FAST TCP. *IEEE Transactions on Automatic Control*, 2008. *To appear.*

Krister Jacobsson, Lachlan L. H. Andrew, Ao Tang, Karl H. Johansson, Håkan Hjalmarsson and Steven H. Low. ACK-clocking dynamics: Modelling the interaction between windows and the network. In *Proceedings of the IEEE Infocom mini-conference 2008*, Phoenix, 2008.

Ao Tang, Lachlan L. H. Andrew, Krister Jacobsson, Karl H. Johansson, Steven H. Low and Håkan Hjalmarsson. Window flow control: Macroscopic properties from microscopic factors. In *Proceedings of the IEEE Infocom 2008*, Phoenix, 2008.

Ao Tang, Krister Jacobsson, Lachlan L. H. Andrew and Steven H. Low. An accurate link model and its application to stability analysis of FAST TCP. In *Proceedings of the IEEE Infocom 2007*, Anchorage, 2007.

Krister Jacobsson, Håkan Hjalmarsson and Niels Möller. ACK-clock dynamics in network congestion control – an inner feedback loop with implications on inelastic flow impact. In *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, 2006.

Ao Tang, Krister Jacobsson, Lachlan L. H. Andrew and Steven H. Low. Linear stability analysis of FAST TCP using a new accurate link model. In *Proceedings of the the 44th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, 2006.

Krister Jacobsson and Håkan Hjalmarsson. Towards accurate congestion control models: validation and stability analysis. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*, Kyoto, 2006.

Niels Möller, Karl H. Johansson and Krister Jacobsson. Stability of window-based queue control with application to mobile terminal download. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*, Kyoto, 2006.

Krister Jacobsson and Håkan Hjalmarsson. Closed loop aspects of fluid flow model identification in congestion control. In *Proceedings of the 14th IFAC Symposium on System Identification*, Newcastle, 2006.

Closely related papers which includes material that is not explicitly covered in the thesis are:

Martin Suchara, Lachlan L. H. Andrew, Ryan Witt, Krister Jacobsson, Bartek P. Wydrowski and Steven H. Low. Implementation of provably stable MaxNet. In *Broadnets 2008*, London, 2008. *Submitted.*

Krister Jacobsson and Håkan Hjalmarsson. Local analysis of structural limitations of network congestion control. In *Proceedings of the 44th IEEE Conference on Decision and Control*, Seville, 2005.

Krister Jacobsson, Håkan Hjalmarsson and Karl H. Johansson. Unbiased bandwidth estimation in communication protocols. In *Proceedings of the 16th IFAC World Congress*, Prague, 2005.

Krister Jacobsson, Niels Möller, Karl H. Johansson and Håkan Hjalmarsson. Some modeling and estimation issues in control of heterogeneous networks. In *Proceedings of the 16th International Symposium on Mathematical Theory of Networks and Systems*, Leuven, 2004.

Chapter 2

Background

RESOLUTION: The Federal Networking Council (FNC) agrees that the following language reflects our definition of the term "Internet". "Internet" refers to the global information system that - (i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein.

The Federal Networking Council, October 24, 1995.

JANUARY 1, 1983 was "flag-day" for the transition of the ARPANET host protocol from the early Network Control Protocol (NCP) to TCP/IP. According to the resolution above this was the day the Internet was born. Subject to exponential growth the Internet has since then grown to be the largest dynamical system ever built by mankind and it has revolutionized the computer and communication world like nothing before. The objective of the first section of this chapter, Section 2.1, is to give a brief introduction to the Internet, and the principles it relies on. After that, in Sections 2.2 and 2.3, we zoom in on the congestion control mechanisms whose robust and scalable designs clearly have been instrumental to the Internet revolution. The remainder of the chapter, Sections 2.4 and 2.5, outlines recent analytical tools devoted to increase the understanding of these feedback control algorithms.

2.1 The Internet and IP networking

The Internet is a worldwide system of interconnected computer networks, a network of networks, linked by copper wires, fiber-optic cables, wireless links etc., that communicates using *packet switching* and the standard *Internet Protocol* (IP).

The Internet provides the transport infrastructure for a wide range of services such as, e.g., electronic mail, file sharing, telephony, radio and television as well as other types of streaming media. It also hosts the *World Wide Web* (WWW), a virtual network of interlinked *hypertext* documents. This "web" of information, not to be considered synonymous with the Internet, is hand in hand with the Internet ubiquitous in societies in the developed world today.

Much has been written about the history, the technology, and the usage of the Internet. The material covered in this very brief discussion on TCP/IP can be found in, e.g., (Leiner *et al.*, 1997), (Hunt, 1998) and (kc claffy *et al.*, 2007).

2.1.1 The road to TCP/IP

In 1965 MIT researcher Lawrence G. Roberts together with Thomas Merrill connected the TX-2 computer in Cambridge, Massachusetts, to the Q-32 in Santa Monica, California, using a low speed dial-up telephone line. This was the first *Wide-area computer network* (WAN) ever built. The result of the experiment was the observation that time-shared computers worked well together, but that the circuit switched telephone network did not meet the demands. They realized that packet switching was the route to follow.

In a packet switched network packets of data are sharing links with other traffic. Non-exclusive access to circuits is normative, however to the price of no user end-to-end capacity guarantees. Data packets are rather delivered on a "best-effort" basis. Each carrier in the network is doing its best such that packets are reaching their destination. Packet switching is fundamentally different from public switched telephone networks that uses circuit switching to allocate an exclusive path with a predefined capacity across the network for the full duration of the connection. The resource sharing model of packet switching was chosen since it is the far more economically efficient way to utilize existing networking resources.

The first published work on packet switching theory appeared in 1961 and was authored by Leonard Kleinrock at MIT. Breaking the theoretical ground, Kleinrock's Network Measurement Center at UCLA in 1969 became one of four network nodes of the world's first packet switched network—the ARPANET. More and more computers were added to the network during the following years. At the same time as the network grew quickly the effort on completing the early used host-to-host protocol, the Network Control Protocol (NCP), and other network software continued. The work on NCP was finally finished in 1972. This was also the year when electronic mail was introduced.

At this stage the ARPANET started to turn into the Internet by assimilating other networks like satellite networks and ground-based packet radio networks. The organic growth was enabled by a meta-level "Internetworking architecture" allowing connections of individual networks based on different technologies. This open architecture networking is the key technical idea behind the Internet. The concept was initially introduced by Robert E. Kahn in 1972 who developed a new version of the NCP protocol suitable for an open-architecture network environment. The NCP protocol was designed for the ARPANET exclusively and relied on it to provide end-to-end reliability, furthermore it could not address computers in other networks. Kahn's refined protocol solving these issues eventually was called the *Transmission Control Protocol/Internet Protocol* (TCP/IP).

As the Internet continued to evolve, TCP/IP did as well. From experimental work on Voice over IP (VoIP) is was realized that while the protocol worked fine for file transfer and remote login applications, it was less suitable for delay sensitive applications. Finally, this led to a separation of the protocol into the simple IP responsible for addressing and forwarding of packets, and TCP which provided flow control and packet loss recovery. To meet the demands of applications more sensitive to delays than packet loss, the *User Datagram Protocol* (UDP) was introduced in 1980 as a means to access the basic service of IP.

After several years of planning January 1, 1983 was the official transition day of the ARPANET host protocol from NCP to TCP/IP. Soon after that the Internet was a well established technology used by a large number of researcher and developers, by now it also becomes more and more popular among other communities. The dramatic growth of the Internet had just begun, it now started to double its size every year.

2.1.2 End-to-end principle and protocol layering

The robustness and the adaptivity of the Internet's architecture has enabled its explosive growth as well as explosive innovation in edge services and applications. It uses a design based on an end-to-end model of connectivity, initially discussed in the paper (Saltzer *et al.*, 1984). The rationale is to strive to put intelligence and state maintenance at the edges, while keeping the network core simple. The basic idea of the end-to-end principle is to assign different functions to *layers*, or subsystems, of the system. Such layerings are desirable to enhance modularity. As stated by Saltzer *et al.* (1984) end-to-end processing tends to increase reliability, and thus robustness.

In a layered architecture, designers only need to deal with the complexity of a single layer, which facilitates development of new functionalities/services. In a communication network context, e.g., the network is just a neutral transport medium and does not have to know what applications are running on it. An application developer, e.g., can deploy new applications without any permission from the Internet Service Providers (ISPs) or any extra charge on the normal service fees.

The Open Systems Interconnection Basic Reference Model (OSI model) defines how functions should be assigned in a generic communication system. The abstraction specifies seven layers. The model is illustrated to the left in Figure 2.1. The layers are are from top to bottom: Application layer, Presentation layer, Session layer, Transport layer, Network layer, Link layer and Physical layer. Each layer provides services to the layer above, and receives services from the layer below. The Application layer consists of the application programs that use the network. The

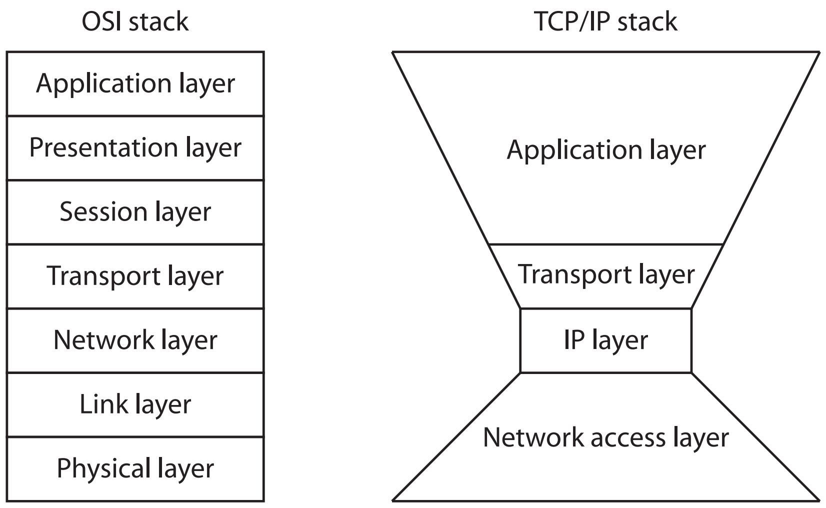


Figure 2.1: Left: The OSI networking stack. Right: The IP networking model.

Presentation layer standardizes data presentation to the applications. The Session layer manages sessions between applications. The Transport layer provides end-to-end reliability, error detection as well as correction. The Network layer manages connections across the network for the layers above. The Data link layer delivers data across the physical link. The Physical layer defines the physical characteristics of the network media.

Note that a layer does not define a single protocol, but a function that may be performed by different protocols. Each layer, thus, may contain several protocols providing services within the scope of that layer. As an example, a file transfer protocol and an electronic mail protocol do both provide user services, moreover both are part of the Application layer.

2.1.3 TCP/IP

The structure of the Internet is defined by the TCP/IP model. The layering is according to the right picture of Figure 2.1. The three uppermost layers of the OSI model are lumped into a single layer, the Application layer, in the TCP/IP model. The IP layer, sometimes called the Internet layer, of the TCP/IP model corresponds to the Network layer in the OSI model. Furthermore the two lower layers in the OSI model are represented by the single Network access layer in TCP/IP.

The heart of TCP/IP is the Internet Protocol (IP). IP communicates data across a packet switched internetwork. An internetwork is a network segment that consists of two or more smaller network segments, a “network of networks”. Recall that in a packet switched network, packets of data are sharing links with other traffic. Briefly, data from a transport layer protocol is divided into chunks and packetized together with destination (IP) address. IP then routes the packets through the network to the final destination. Being a lower layer protocol, IP only provides best effort packet delivery between nodes in line with the end-to-end principle to keep the core simple. Thus packets may, e.g., be corrupted, arrive out of order, or even be lost or dropped/discarded along the route. The main feature of IP is that it can be used over heterogeneous networks. The physical network could consist of any mix of, e.g., ethernet, Wi-Fi or cellular networks, providing packet transport between nodes sharing the same link. The hourglass shape of the TCP/IP networking model to the right in Figure 2.1 illustrates that IP is used for packet transportation alone on the Internet, while there are many different network access technologies as well as numerous transport layer protocols, and even more application protocols.

Next, let us work our way up through the layers from bottom to top.

The Network access layer

The lowest layer in the TCP/IP suite is the Network access layer. The protocols in this layer provide upper layers with data delivery to other devices on a directly attached network. The layer defines how to use the network to transmit an IP datagram (packet). It converts the IP address into an address that is understandable for the specific physical network that is used. Each hardware technology demands its own Network access protocol.

The IP layer

The IP layer is sometimes referred to as the Internet layer and is the layer above the Network access layer. The Internet Protocol is without doubt the most important protocol in the IP layer. All protocols in layers above use IP as data delivery service. By calling the Network access layer, IP ties networks based on different kinds of physical media together into one which forms the Internet. All incoming and outgoing data flows through IP, regardless of its final destination.

The IP is the building block of the Internet and it includes the following functions:

- Defining the datagram, i.e., the packet, which is the basic unit of transmission in the Internet.
- Defining the Internet addressing scheme.
- Moving data between the Network access layer and the Transport layer.
- Routing datagrams to remote hosts.
- Performing fragmentation and re-assembly of datagrams.

IP is reliable in the sense that it accurately delivers data according to the specified address. However, IP does not check if this data was correctly received. This end-to-end functionality is left for the layer above.

The Transport layer

In the TCP/IP protocol stack the Transport layer is responsible for the end-to-end data transfer, i.e., that the intended information is delivered to the receiver. It is the first layer that offers reliability.

The by far most used transport layer protocol on the Internet today is TCP, offering end-to-end error detection and correction. Basically TCP detects corrupted or lost segments of packets, and resends them until the data is successfully received.

Such end-to-end delivery guarantee may, however, not be suitable for all applications. Some applications, e.g., Voice over IP (VoIP), are more sensitive to delay than to dropped packets. Hence best effort protocols are a more appropriate choice for such cases. The User Datagram Protocol (UDP) is currently the most used best effort protocol. It is typically used for streaming media, i.e., audio, video, etc., but also simple query/lookup applications.

The Application layer

The top layer of the TCP/IP stack is the Application layer. It includes all processes that use the Transport layer to deliver data to the IP layer. There is a wide range of application protocols. Most of them provide user services, but some of them are directly behind applications. As the innovation in edge applications continues new services and protocols are added to this layer. Examples of widely used Application layer protocols are:

- The Network Terminal Protocol (Telnet), provides text communication for remote login and communication across the network.
- The File Transfer Protocol (FTP), downloads and uploads files across the network.
- The Simple Mail Transfer Protocol (SMTP), delivers electronic mail messages across the network.
- The Hyper Text Transfer Protocol (HTTP), used by the World-Wide-Web to exchange text, pictures, sounds, and other multi-media information via a graphical user interface.

Some protocols can only be used if the user has some knowledge about the network while others may run without the users' awareness of their existence.

2.2 The Transmission Control Protocol

We have learned that end-to-end reliability in TCP/IP is managed by the Transport layer. The topic of this section is the Transmission Control Protocol (TCP) which is the predominant transport protocol of the Internet today carrying more than 80 % of the total traffic volume (Fomenkov *et al.*, 2004). TCP is used by a range of different applications such as web-traffic (WWW), file transfer (FTP, SSH), and even streaming media application with "almost real-time" constraints such as, e.g., Internet radio.

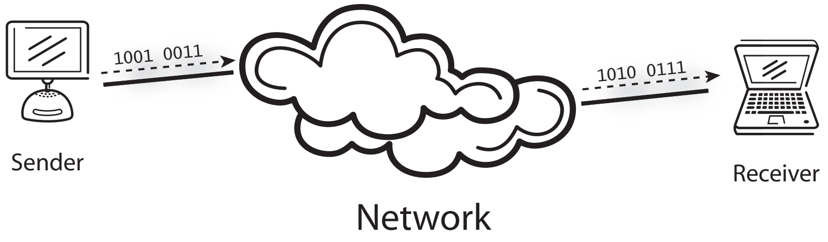


Figure 2.2: A sender transfers data to a receiver after establishing a connection with a three-way handshake. It is also a data flow from the receiver to the sender since received data is acknowledged.

TCP is a connection-oriented protocol. This means that it establishes an end-to-end connection between two hosts that are communicating. Assume host A wants to transfer data to host B, then we will refer to host A as the *sender* and host B as the *receiver*, see Figure 2.2. The *connection* or *flow* is defined by the sender/receiver pair. We will sometimes equivalently use the term *source* to denote the sender. The *path* is the subset of links in the network that is utilized by the flow. The procedure of setting up a connection in a connection-oriented protocol involves the exchange of control information, this is referred to as a *handshake*. A handshake of TCP takes three exchanged segments (packets) and is thus called a *three-way handshake*.

2.2.1 Positive acknowledgment with re-transmission

The packet transfer reliability of TCP is accomplished by a mechanism called Positive Acknowledgment with Re-transmission (PAR). Shortly, in a system using PAR the receiver re-sends data according to some rule unless it hears from the receiver that the data was successfully received. In a connection using PAR the data flow is bi-directional since the ACKs flow from the receiver to the sender.

In TCP each unit of data, i.e., a packet, contains a checksum such that the receiver can verify if the received data is correct. If the packet is successfully received the receiver sends an acknowledgment packet (ACK) back to the sender to verify this—a positive acknowledgment. If the received data is faulty, the receiver just discards it. To compensate for damaged or lost packets, the TCP sender re-transmits each packet that has not been acknowledged after an appropriate time-out period.

TCP uses data sequence numbering to identify packets, and cumulative ACKs to inform the sender. In a cumulative ACK strategy, an ACK referring to a particular point within a data stream implicitly acknowledges all data up to the ACKed sequence number. This is in contrast to per-packet ACKs which only acknowledges the packet with the very same sequence number as the ACK itself.

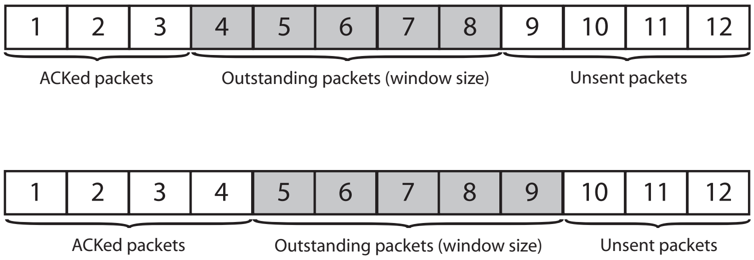


Figure 2.3: The sliding-window mechanism. In this example the number of outstanding packets, the (send) window size, is set constant to 5 packets. In the top figure the sender has transmitted 8 packets, whereby 3 has been acknowledged. In the lower figure one more packet has been acknowledged. Thus the subsequent packet to be transmitted, packet number 9, is sent and the window "slides" forward one packet on the data stack.

2.2.2 Flow control

To avoid that the sender transmits data too fast for the receiver to reliably receive and process it, TCP implements an end-to-end flow control. Such a mechanism is necessary in a heterogeneous network environment where end terminals have varying processing capabilities. In, e.g., a scenario of a cell phone downloading a file from a network server, the server is likely to have much better processing capacity than the processor in the cell phone and thus would overflow the cell phone protocol software in the case of no flow control.

The sliding-window

The flow control in TCP is accomplished by a *sliding-window* mechanism. The receiver controls the data flow from the sender by telling the sender how much data it can buffer, this is known as the *receive window* size. This information is communicated from the receiver to the sender via the ACKs. The sender can only send up to this amount of data before waiting for next ACK packet. This is controlled by that the sender maintains a *send window*, corresponding to the number of packets transmitted but not yet acknowledged, that has to be equal or smaller than the receive window size advertised by the receiver. Figure 2.3 illustrates how a constant send window size "slides" forward as an ACK is received, thereby the name sliding-window mechanism.

Since the focus of this work is congestion control, it will throughout the thesis be assumed that the receive window is larger than the send window. Thus the receive window can be ignored and the short-hand term *window* will be used for the send window.

2.2.3 Congestion control

While flow control originally was introduced as a means for the receiver to adjust the sender sending rate, the mechanism does not explicitly consider the state of the network or addresses how network resources should be distributed among users (the issue of fairness). To solve this, TCP apply congestion control in the form of dynamical adjustment of the window size.

Network routers typically have packet queues to hold packets instead of discarding them when the network is busy. However, since a router has limited resources the size of this queue is limited as well. When the network is congested queues will start to build up, and thus the delay, and at some point packets must be dropped. This is, in various ways (depending on the precise TCP algorithm), utilized by the TCP congestion control.

In TCP the sender updates the window size according to some rule using an estimate of the state of congestion in the network as input. The information about the network state is carried to the sender by the returning ACKs. The congestion estimation must be performed implicitly by, e.g., detecting lost packets or monitoring queuing delay, or any other explicit signal correlated with network congestion.

Macroscopically, the difference between various TCP “flavors” is the way estimation is performed and the rule the window size is updated according to. Before giving an overview of different versions of TCP in Section 2.2.4, let us return to the sliding-window mechanism since it turns out that, in fact, it is a congestion control mechanism in itself.

ACK-clocking

The sliding-window mechanism controls the maximum amount of data a TCP sender can keep inside a network. In contrast to this explicitly controlled data load, the effective sending rate of a sender is dependent on the dynamics of the network itself.

Recall that the sliding-window mechanism only transmits a packet at the arrival of an ACK and when the size of the window allows so, see Figure 2.3 for the case of a constant window. This implies that the sending rate is auto-regulated, or “clocked” by the rate of the received ACKs. We will refer to this as *ACK-clocking* in the sequel. If a packet experiences congestion it will be delayed due to that queues are building up, therefore the ACK rate and the sending rate, and consequently the load on the network is decreased accordingly.

Even though, as we will see in Chapter 4, the ACK-clocking has stabilizing properties in itself, it does not provide fair and efficient utilization of resources. Consider, e.g., a sender with constant window size and a bandwidth delay product less than the capacity. Note that the senders sending rate is less than the capacity for this case and that the link thus will remain under utilized if no other traffic is present. This motivates the dynamical adjustment of the window size used in TCP.

We have seen that the sender controls the sending rate as a function of the rate

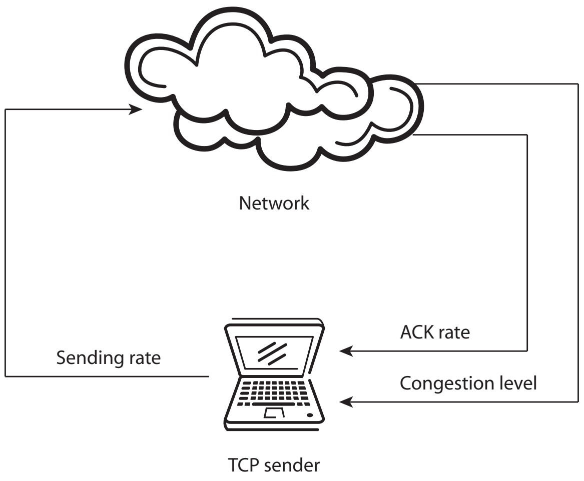


Figure 2.4: TCP congestion control in a sender perspective. The TCP sender transmits data to the TCP receiver located somewhere in the network. The sending rate adapts on the rate of the received ACKs due to ACK-clocking and the level of congestion in the network via the window control.

of the returning ACKs and the window size, and that the window size is updated according to the level of congestion of the network (and that this information is communicated to the sender by the ACKs). From this it is realized that from a signal based point of view the congestion control mechanism in TCP constitutes two feedback loops according to Figure 2.4. It turns out that this perspective will be useful when modeling the network, see Chapter 3.

2.2.4 TCP overview

In the original TCP specification (RFC 793) window based flow control was specified as a means for the receiver to regulate the amount of data sent by the sender. This to prevent data overflow at the receiver side. The specification did not, however, include dynamic adjustment of the congestion window size in response to congestion.

Beginning in October 1986 the Internet suffered from a series of congestion collapses, events where the network operates in a stable but degraded condition where throughput is several orders of magnitude lower than the capacity (Nagle, 1984). This eventually led to that the congestion avoidance mechanisms that are now required in TCP implementations were developed in 1988 by Van Jacobson (Jacobson, 1988; Allman *et al.*, 1999; Floyd, 2000). The resulting implementation of Jacobson's work, the TCP Tahoe release of BSD Unix, reacts to lost packets as a congestion signal fed back from the network to the sender. This is still what prevents congestion collapse of today's Internet.

The research effort on congestion control has been considerable after 1988. However, the most widely deployed TCPs of today, TCP Reno (Stevens, 1997) and TCP NewReno (Floyd *et al.*, 2004) (c.f. TCP SACK option (Mathis *et al.*, 1996)), are in principal similar to their predecessor TCP Tahoe, yet improved versions equipped with refined congestion control functionality. It is clear that Van Jacobson's original design was extremely well-founded and strongly has contributed to that the Internet has been able to grow exponentially. However, it is also well known by now that TCP Reno's performance degrades as networks scale up in size and capacity (Hollot *et al.*, 2002; Floyd, 2003; Low *et al.*, 2003). Standard TCP Reno performs reasonable across a wide variety of network conditions, though it is not optimized for any. However there are alternative more specific protocol solutions, enhanced for specialized scenarios. We will here briefly review the standard TCP NewReno and some of the most well known TCP proposals. They are classified by the type of congestion feedback on which they rely on.

Loss-based protocols

The rationale of loss-based TCP protocol is that packet loss is an indication of network congestion. If such an event is detected, the missing data has to be retransmitted of course, but also the sending rate should be adjusted to reduce the load on the network and thus the probability of further packet loss.

Loss-based TCPs probes the network to assess the available bandwidth, tries to maintain a sending rate that matches this value and if the available bandwidth is changed adapts accordingly. In broad outlines, this is done by ramping up the window size, and thus the sending rate, until the network gets congested and queues build up and eventually saturate which implies that packets are dropped (assuming a first-in-first-out/drop-tail environment). When this occur the TCP sender backs-off by decreasing the window to lower its load on the network before ramping up the rate again to once again probe for the available bandwidth. Loss-based TCPs strive towards saturating network buffers. The motivation for this is that a link with a non-zero queue is operating at full capacity and thus the network resource is efficiently utilized.

Packet loss detection is obviously important in TCP. This is done with the means of *duplicate* ACKs. Recall that TCP uses cumulative ACKs, acknowledging all packets up to its sequence number. If an ACK arriving at the sender has the same sequence number as a prior received ACK it is said to be duplicate. This is illustrated in Figure 2.5. When this occurs a packet may have been lost or delayed and delivered out-of-order; it is also a possibility that the ACK has been duplicated by the network. In, e.g., TCP NewReno a packet is considered lost if: no ACK is received for that packet within a certain time-out period; or, if three duplicate ACKs are received. Such congestion events triggers TCP window control to take action.

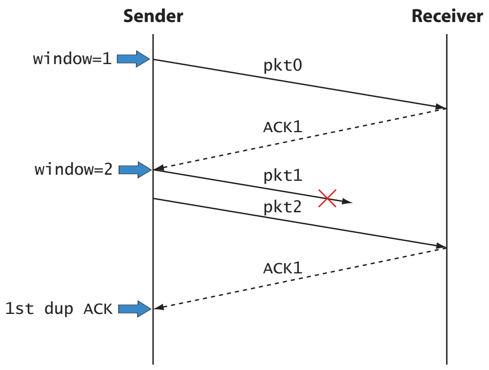


Figure 2.5: Duplicate ACK illustration. The receiver acknowledges the next expected packet in the data sequence.

Standard TCP (TCP NewReno) The principal operation of TCP NewReno involves three main mechanisms: *slow start*, *congestion avoidance* and *fast retransmit/fast recovery.*

At the start-up of a connection the sender starts cautiously with a window size of just a few packets, it then tries to quickly identify the bottleneck capacity by increasing the window size exponentially. This is achieved by incrementing the window with one packet per received ACK, and thus the window is doubled every RTT. This mode is called slow start, see Figure 2.6.

The system remains in slow start until the window reaches a threshold, the *Slow start threshold*, where the system enters congestion avoidance. During congestion avoidance the window is increased more cautiously with the multiplicative inverse of the window for each received ACK. This results in that the window increment is one packet every RTT.

If the sender detects a packet loss by receiving three duplicate ACKs, it retransmits the lost packets, halves the window, and re-enters congestion avoidance. This is referred to as fast retransmit/fast recovery. This is in contrast to when a loss is detected through a time-out. In that case, after the packets have been transmitted, the window is re-initialized and and the sender re-enters slow start instead of congestion avoidance.

Remark that the effective sending rate of the protocol is not determined by the window size alone. Due to the ACK-clocking mechanism it is proportional to the window size divided by the round trip time. A senders sending rate is thus affected by the state of the network through the queuing delay packets (and ACKs) are subject to along the path, see Section 3.3. This means that a doubled window size does not corresponds to a doubled sending rate in a congested network.

In steady-state operation under normal conditions, TCP remains in a cycle alternating between congestion avoidance and fast retransmit/fast recovery. It thus

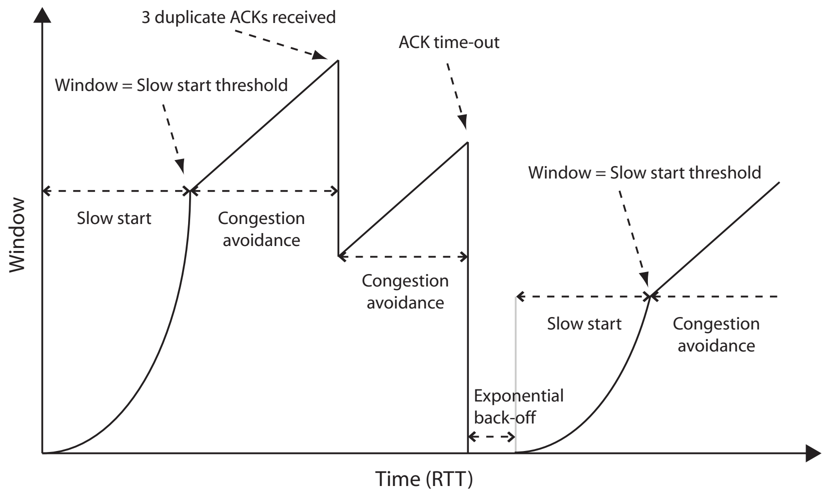


Figure 2.6: Idealized TCP Reno/NewReno behavior.

sometimes increases the window additively and sometimes reduces the window with a multiplicative factor equal to a half. This strategy to dynamically adapt the window size to the amount of congestion in a network was proposed by Jacobson (1988) and is called additive-increase-multiplicative-decrease (AIMD). In general AIMD the window is increased with α packets per RTT until a packet loss is detected whence the window is decreased a multiplicative factor β. This is achieved by increasing the window `w` with 1/`w` at the arrival of an ACK, and by reducing it with β`w` at a loss. In pseudo code, the window update is

$$\begin{aligned} \texttt{Ack}: \quad & \texttt{w} \leftarrow \texttt{w} + \frac{\alpha}{\texttt{w}}, \\ \texttt{Loss}: \quad & \texttt{w} \leftarrow \texttt{w} - \beta\texttt{w}. \end{aligned}$$

From the previous discussion it is realized that TCP NewReno applies AIMD with parameters $\alpha = 1$ and $\beta = 1/2$.

An illustration of the window evolution of TCP NewReno is shown in Figure 2.6. At the startup the window is increased exponentially using Slow start. It remains in this state until the window equals the Slow start threshold where it enters Congestion avoidance where the window is increased linearly. While in this mode, eventually three duplicate ACKs are received due to a lost packet. The window is then halved and increased linearly again. The next event is an ACK time-out, i.e., no ACK is received for a packet within a certain time-out period. The protocol interprets this as that the network is severely congested and thus remains silent for a while and then re-starts the bandwidth probing in Slow start. When the window grows beyond the Slow start threshold the Congestion avoidance mode is entered.

Note that the Slow start threshold has changed from the initial cycle. At a loss event it is typically set to half the window size when the loss occurred.

While the AIMD control strategy of TCP NewReno and its predecessors historically has shown to perform quite well over a wide range of networks it is well known that its performance degrades over wireless networks, where packet loss may not be due to congestion, and in networks with large bandwidth-delay products. Another issue that has received much attention in the networking community is that TCP NewReno punishes flows with large delays, a larger delay implies a smaller share of the available bandwidth in steady state.

HighSpeed TCP Already in Chapter 1 an example was presented that showed that the standard AIMD control strategy used by TCP is not aggressive enough to be able to fully utilize high speed networks. The rationale of HighSpeed TCP (Floyd, 2003) is to change the control in such environments but keep the standard TCP behavior in environments with heavy congestion—this to prevent any new risks of congestion collapse. This is achieved by adapting the parameters of the AIMD algorithm to the window state. We say that HighSpeed TCP uses an AIMD($\alpha(\mathtt{w}), \beta(\mathtt{w})$) update strategy.

Using an engine analogy one can view HighSpeed TCP as a turbo charged version of standard TCP; the higher throughput the protocol is operating with, the higher the turbo charged boost to the normal performance. For normal conditions the congestion avoidance function is unaltered, but when the packet loss rate falls below a threshold value the higher speed congestion avoidance algorithm kicks-in. For standard TCP the average steady-state rate in packets per RTT, w, as a function of the packet-loss rate, q, is roughly

$$w = 1.2/\sqrt{q},$$

(the classic formula will be derived later in Example 2.4.1). This response function is plotted as the solid line in Figure 2.7. The default higher-speed response function proposed by HighSpeed TCP is given by

$$w = 0.12/q^{0.835},$$

designed to achieve a transfer rate of 10 Gbit/s over a path with 100 ms round trip delay and a packet loss rate of 1 in 10 million packets. It appears as the dashed line in Figure 2.7. Here we see how HighSpeed TCP preserves the fixed relationship between the logarithm of the sending rate and the logarithm of the packet loss rate, but alters the slope of the function. The steeper slope corresponds to that the protocol departs from the standard AIMD control and increases its congestion avoidance window increments as the packet loss falls.

Scalable TCP Yet another high-speed TCP proposal is Scalable TCP (Kelly, 2003b). In principle Scalable TCP uses an AIMD($\alpha(\mathtt{w}), \beta$) type of window update mechanism with parameters $\alpha = a\mathtt{w}$ and $\beta = b$. This results in that the window is inflated by a constant value a for each received ACK, and at a congestion event

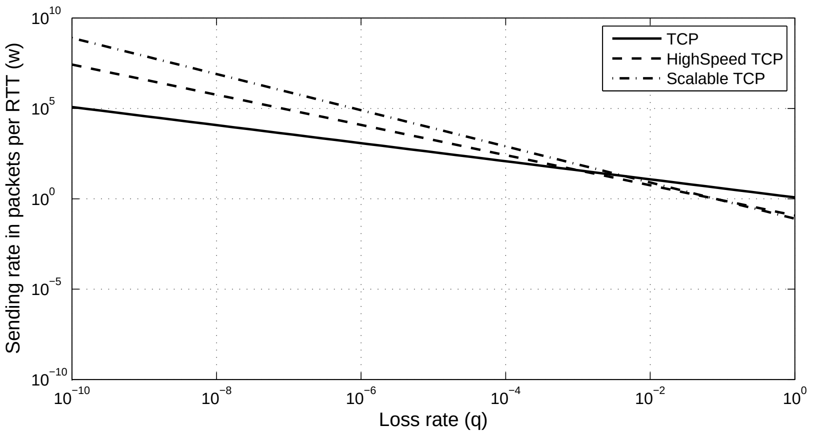


Figure 2.7: TCP response functions.

the window is reduced by the fraction b. The probing is in this way decoupled from the RTT interval. This implies that connections with longer network paths exhibit similar behavior as connections with shorter paths. The response function of Scalable TCP is

$$w = \frac{a}{bq}.$$

It is plotted for default protocol parameters $a = 0.01$ and $b = 0.125$ as the dashed-dotted line in Figure 2.7. The linear relationship between the logarithm of the packet loss rate and the logarithm of the sending rate is preserved. The greater slope indicates that Scalable TCP uses an even more aggressive multiplicative increase rule in the congestion window than HighSpeed TCP.

BIC and CUBIC A slightly different approach to window based congestion control compared to previously described algorithms is taken in CUBIC (Rhee and Xu, 2005) and its predecessor BIC (Xu *et al.*, 2004). Network congestion control is in these algorithm viewed as a search problem where the protocol tries to find the threshold rate triggering packet loss which is the binary (“Lost”/”Not lost”) feedback provided by the system. BIC uses a binary search technique to set the target window size to the midpoint between the maximum window size $W_{\max}$, the largest window size where packet loss occur, and the minimum window size $W_{\min}$, the window size where the flow does not see any packet loss. The result is a hybrid window increment function which is linear initially but slows down as it approaches the previous point where packet loss occurred.

It has been shown empirically that BIC may be too aggressive compared to standard TCP, in particular in small RTT environments or low speed situations. CUBIC tries to solve these issues by inflating the window according to a third-order polynomial in absolute time, the window function is shown in Figure 2.8.

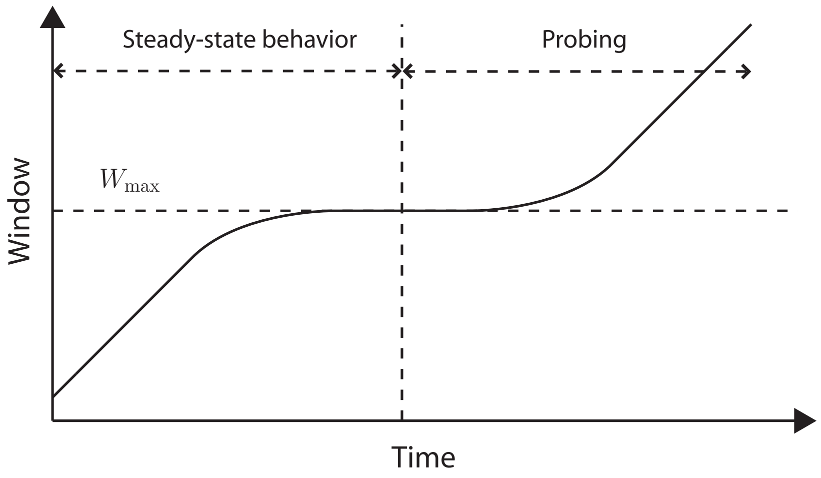


Figure 2.8: The window growth function of CUBIC.

The window growth function of CUBIC mimics and is qualitatively the same as the BIC window growth function, it is however flatter close to $W_{\max}$ were the previous packet drop was encountered. This more careful control makes CUBIC more stable than BIC. Furthermore since the growth function is independent of the RTT, steady-state throughput is that as well. While CUBIC is a high-speed version of TCP it is less aggressive at low packet-loss rates compared to Scalable TCP and HighSpeed TCP. This makes it TCP friendlier under moderate high speed conditions where standard TCP is still usable.

H-TCP The design objective of H-TCP (Leith and Shorten, 2004) is to improve bandwidth-delay product scaling, mitigating RTT unfairness as well as decouple throughput efficiency from network buffer provisioning with smallest possible changes to standard TCP. The guiding rationale is that TCP has proved to be remarkably effective and robust as network congestion controller and it is therefore motivated to retain as many of it features as possible.

For good performance in high bandwidth-delay networks the aggressiveness of the protocol may be increased with larger windows. This, however, implies that flows with large windows are more aggressive than flows with smaller windows. As a result flows with small windows have difficulties in maintain their share of the resources, e.g., in a startup scenario. H-TCP solves this issue by proposing a timer-based window growth function, where for an initial period the standard additive increase is preserved, but after the end of this period the window is inflated according to a second-order polynomial in the time elapsed since the last congestion event. At a packet loss the window is reduced with a multiplicative factor that is adapted to the variance of the RTT interval. A typical window evolution epoch

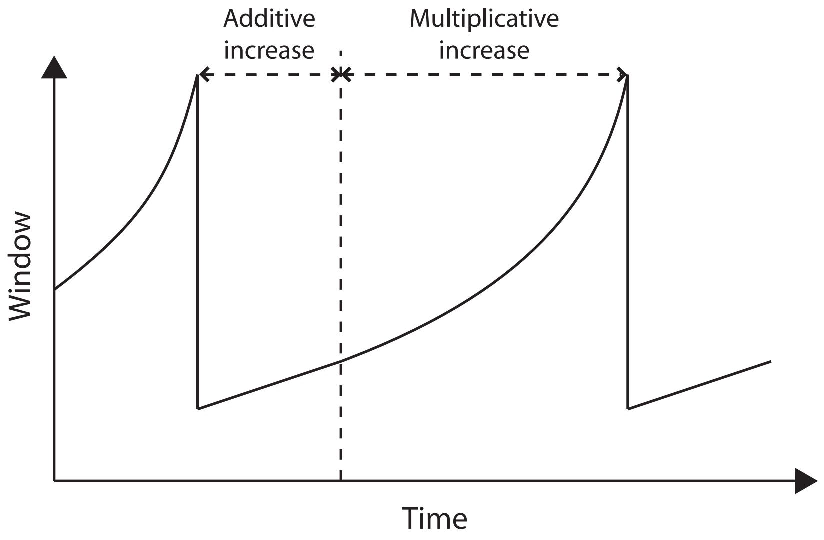


Figure 2.9: Window size evolution of H-TCP.

is shown in Figure 2.9 where the two phases of increase are shown as well. This strategy retains the fundamental properties of standard TCP (Shorten *et al.*, 2007) at the same time as having a response function similar to HighSpeed TCP.

Delay-based protocols

The size of the queues in a network is a function of the load on the network. Thus, in the phase of congestion queues are building up. This correlation is utilized by delay-based protocols.

Packet loss is a binary congestion signal. Subsequently each packet loss measurement provides a single bit of information—a packet is dropped or not dropped. This is in contrast to queuing delay measurements which provides multi-bit information, the rate of change of a network queue is proportional to the excess rate over the link. As a consequence, delay-based congestion control can stabilize a network in a steady-state with a target fairness and high utilization. While a loss-based protocol strives towards saturating queues and thus operates under heavy congestion, a delay-based approach may operate at a point of small queues, and thus full utilization, and no packet loss at all. This implies lower steady-state delays which is beneficial for delay sensitive applications.

The fundamentally different modes of operation dramatically reduces the performance of delayed-based protocols in a mixed environment where it co-exist with loss-based algorithms. Thus backward compatibility with standard TCP is an issue. Another challenge in a queuing-delay approach to congestion control is the estimation of the queuing delay. Round trip time samples are available by timing of the returning ACKs. These measurements, however, are normally affected by noise since traffic, and thus queues, is bursty by nature. More crucially, round trip time samples include the physical propagation delay. This quantity is typically unknown to the sender in a congested network. Consequently an unbiased queuing delay estimate may be hard to achieve.

Early proposals of delay-based congestion control are, e.g., (Jain, 1989) and (Wang and Crowcroft, 1992). Widely known implementations are TCP Vegas (Brakmo *et al.*, 1994) and the modern high-speed version FAST TCP (Wei *et al.*, 2006).

TCP Vegas The congestion avoidance design of TCP Vegas exploits the simple idea that, in a congestion control algorithm applying a sender window, the number of bytes in transit is proportional to the expected throughput, and therefore, as the window increases the throughput of the connection should increase as well.

The strategy of TCP Vegas is to adjust the window size to control the amount of "extra" data the connection has in transit, whereby extra data means the data that has to be buffered by the network and would not have been sent if the sending rate exactly matched the available bandwidth.

Let `BaseRTT` denote a given connection's propagation delay, i.e., the RTT of a packet when the network is not congested and queues are empty. TCP Vegas defines the expected throughput by

$$\texttt{Expected} = \frac{\texttt{w}}{\texttt{BaseRTT}}$$

where `w` is the size of the sender congestion window.

Next, TCP Vegas calculates the actual achieved sending rate `Actual` by dividing the total amount of data transmitted between a packet is sent and the corresponding ACK is received with the time interval between these two events, i.e., the RTT.

The window is then adjusted depending on the difference between the expected throughput and the actual sending rate, $\texttt{Diff} = \texttt{Expected} - \texttt{Actual}$. Two threshold parameters are defined, $\alpha < \beta$, in units of kilobyte/s, and when

- $\texttt{Diff} < \alpha$, the window is increased linearly during the next RTT;
- $\texttt{Diff} > \beta$, the window is decreased linearly during the next RTT;
- $\alpha < \texttt{Diff} < \beta$, the window is unchanged.

The simple interpretation of this algorithm is that TCP Vegas adjusts its rate such that the actual rate remains between α and β KB/s lower than the expected rate.

A more instructive interpretation may instead be in terms of buffered packets rather than rates. The total backlog buffered in a path is equal to the window size minus the bandwidth-delay product, i.e., $\texttt{w} - \texttt{Actual} \times \texttt{BaseRTT}$. Note that this is exactly what we get if we multiply `Diff` with the propagation delay `BaseRTT`. Thus multiplying the conditional in TCP Vegas control strategy, i.e., $\texttt{Diff} < \alpha$ and so on, by the propagation delay `BaseRTT` it is realized that a sender adapts its rate to a buffer between $\alpha\texttt{BaseRTT}$ (typically, 1) and $\beta\texttt{BaseRTT}$ (typically, 3) number of packets in it its path (Low *et al.*, 2002c).

FAST TCP Using an equation based design approach FAST TCP is developed to be a high-speed version of TCP Vegas tailored to perform in networks with high-bandwidth delay product. A key departure from traditional TCP design is that FAST TCP uses the same window update law independent of the state of the sender. Under normal operation (i.e., when no packet losses occur) the FAST TCP algorithm updates the window size, `w`, once per RTT according to

$$\texttt{w} \leftarrow (1-\gamma)\texttt{w} + \gamma\left(\frac{\texttt{BaseRTT}}{\texttt{RTT}}\texttt{w} + \alpha\right), \tag{2.1}$$

where `BaseRTT` is the minimum observed RTT, `RTT` is the averaged RTT created by summing an estimate of the propagation delay with a queuing delay estimate obtained by low-pass filtering the observed queuing delay, and $\gamma \in (0, 1]$ and $\alpha > 0$ are design parameters. The constant γ scales the convergence speed of the window control, while the constant α corresponds to the number of packets a flow tries to keep buffered in the network, cf., the threshold parameters of TCP Vegas.

A key feature of FAST TCP is that it maintains a stable equilibrium. This is in contrast to, e.g., TCP NewReno which rather oscillates in a steady-state limit cycle. From the update law (2.1) we get by equating the right-hand and left-hand side that the FAST TCP equilibrium condition is

$$\gamma\left(-\texttt{w} + \frac{\texttt{BaseRTT}}{\texttt{RTT}}\texttt{w} + \alpha\right) = \gamma\left((-\texttt{RTT} - \texttt{BaseRTT})\frac{\texttt{w}}{\texttt{RTT}} + \alpha\right) = 0.$$

Subsequently

$$\frac{\texttt{w}}{\texttt{RTT}} = \frac{\alpha}{\texttt{RTT} - \texttt{BaseRTT}} \tag{2.2}$$

must be fulfilled in equilibrium. The left hand side of this expression, the equilibrium window divided by the equilibrium round trip time, is simply the sender equilibrium sending rate. By noting that the difference $\texttt{RTT} - \texttt{BaseRTT}$ corresponds to the queuing delay it is realized that the equilibrium rate is independent of the propagation delay. Thus, flows sending over the same bottlenecks with the same constant α achieves the same equilibrium throughput. Multiplying both sides of (2.2) with the queuing delay $(\texttt{RTT} - \texttt{BaseRTT})$ we see explicitly that $\alpha = \texttt{w} - (\texttt{w}/\texttt{RTT})\texttt{BaseRTT}$ corresponds to the difference between the window size and the bandwidth-delay product and thus the number of packets backlogged in the path.

The equilibrium condition (2.2) furthermore highlights the importance of accurate knowledge of `BaseRTT`, i.e., the propagation delay in delay-based protocols. A positively biased steady-state estimate of the propagation delay yields an underestimated equilibrium queuing delay. This implies that a sender suffering from biased propagation delay estimates will achieve a larger share of the available bandwidth. This is compensated for by flows with correct knowledge about their propagation delay which must back-off accordingly.

Loss-delay based protocols

The objective of loss-delay based congestion control is to adopt the excellent steady-state throughput properties of protocols using a delay based approach while still being backward compatible with standard TCPs that are loss-based. This is typically handled by unifying the two approaches into one congestion control mechanism with a proactive part reacting on queuing delay and a reactive part probing for the available bandwidth by creating losses. With such a strategy the protocol aggressiveness necessary for efficient utilization of modern high-speed networks can be implemented while the algorithm still can be graceful around maximum sending capacity which the network can accommodate.

TCP-Africa The TCP-Africa algorithm (King *et al.*, 2005) is a hybrid protocol combining the high-speed features of HighSpeed TCP with the ability of sensing congestion of TCP Vegas. It uses the delay metric used in the TCP Vegas algorithm to determine if the bottleneck link is congested. In the absence of congestion it uses the window update rule of HighSpeed TCP which has shown to scale well with high-speed networks. In the presence of congestion, when the delay begins to increase, the protocol switches to the more conservative TCP Reno congestion avoidance rule and increases the window with one packet per RTT. It thus remains slow in inducing congestion events compared to, e.g., HighSpeed TCP. This implies that it can remain compatible with standard TCP flows, at the same time as it can be efficient in utilizing the available bandwidth.

TCP-Africa appears to have improved utilization and resource allocation properties compared to standard TCP. It is also reported to have a resource allocation similar to standard TCP Reno that punishes flows with larger RTT.

Compound TCP Consider an application using two flows, one loss-based and the other delay-based, starting up at the same time over a link with no other traffic. When the link is underutilized, the queue is empty and the aggregated throughput, the sum of the two flows' throughput, is increased as the two flows increase their sending rate. When the queue finally starts to build up the delay-based flow gradually decreases its sending rate while the loss-based flow continues to increase its rate until a packet it lost. Trivially the aggregate throughput during this period is bounded below by the loss-based flow, furthermore the increase rate of the aggregated throughput is gradually reduced.

The main idea of Compound TCP (Tan *et al.*, 2006a) is to behave as the aggregate throughput of a loss-based and a delay-based protocol. It incorporates a scalable delay-based window update component into the standard TCP congestion avoidance algorithm. The window is set to the sum of a window adjusted by the loss-based part and the window controlled by the delay-based part of the algorithm. When the network is underutilized the delay-based part increases its window rapidly, and once the queuing delay is increased it smoothly falls back. As for TCP-Africa, the delay-based part derives from TCP Vegas. It is, furthermore, modified to have scalability comparable to HighSpeed TCP. The conventional loss-based component remains as in the standard TCP congestion avoidance algorithm to achieve backward compatibility.

Compound TCP scales well with increased throughput while still being compatible with standard TCP flows at lower speeds. It also improves network resource allocation with respect to RTT compared to standard TCP.

TCP-Illinois The design of TCP-Illinois (Liu *et al.*, 2006) is based on the belief that an ideal window function should be concave. Intuitively such a shape achieves efficient utilization of resources due to aggressive window increase for smaller window sizes, as well as it avoids heavy congestion due to the more careful increase as the window increases. This is contrast to loss-based high speed protocols as, e.g., Scalable TCP, HighSpeed TCP and H-TCP which rather have convex window functions and thus are most aggressive when they send with their maximum capacity.

The TCP-Illinois design objective is to achieve a concave window function using a general AIMD algorithm. This is realized by adapting the α and β AIMD parameters to the amount of congestion. The window increment parameter α is therefore set to be large when far from congestion and smaller as the amount of congestion increases. For backward compatibility with standard TCP the decrease parameter β is set small when far from congestion and large when close to congestion. To estimate the amount of congestion in the network TCP-Illinois uses queuing delay. Thus loss is used to determine the direction of the window size change, and delay to adjust the pace of the change.

TCP-Illinois is reported to achieve better throughput than standard TCP in high-speed networks, to have an improved network resource allocation as well as being compatible with standard TCP.

Explicit feedback protocols

The design of TCP/IP is based on the end-to-end principle. The rationale is to strive towards keeping the network core simple while putting intelligence at the edges. Advantages are, e.g., increased scalability and robustness of the system. This is, however, achieved to the price of constraints on control performance.

The idea of explicit feedback protocols is to slightly violate the end-to-end principle and incorporate more control intelligence in network routers. By explicitly communicating a measure related to the amount of congestion, routers can more accurately notify senders how to adjust the sending rate. This is typically done via fields in the packet header and the returning ACKs. The congestion control design problem is thus not only to design the sending rate adjustment in the endpoint protocol, but also to find an appropriate congestion measure and an update law to be implemented in the routers. The latter control is referred to as Active Queue Management (AQM), which is the unified name for router techniques that supports edge control, like standard TCP, in preventing network congestion. AQM is further discussed in Section 2.3.

All previously discussed TCPs base their control on implicit congestion information (e.g., dropped packets or queuing delay) that must be estimated and interpreted by the sender. Examples of protocols using explicit multi-bit feedback communicated from the network routers to the endpoints via fields in the packet header are XCP (Katabi *et al.*, 2002), RCP (Dukkipati and McKeown, 2006), MaxNet (Suchara *et al.*, 2008) and JetMax (Zhang *et al.*, 2006). Naturally, compared to standard TCP, the performance of all these protocols are superior in terms of their intended use. Implementation issues and backward compatibility is rather the main challenge for this type of protocols.

2.3 Active Queue Management

The control performance that can be achieved from the edges of a network is limited. To support the endpoint congestion avoidance mechanisms, congestion detection and control in the routers may be introduced as a complement. Such mechanisms inside the network are commonly called Active Queue Management.

2.3.1 The need for Active Queue Management

The classical way for managing router queues is to accept incoming packets until a pre-defined maximum length is reached, and then discard all arriving traffic until the queue has decreased due to that a packet from the queue has been transmitted. This queue management algorithm is called *drop tail*, or tail drop, since the tail of the sequence of packets are dropped. Drop-tail management has been used by Internet routers for years, though is has two drawbacks when operating in an environment with loss-based standard TCP, namely *lock-out* and *full queues* (Braden *et al.*, 1998).

1. **Lock-out**
   In a lock-out situation a single or a few flows are occupying the whole queue space, thus preventing other connections getting access to the queue. This phenomenon is often attributed to flow synchronization or other timing effects.
2. **Full queues**
   The objective of maintaining data buffers for a router is to be able to absorb bursts of data. This is essential in a packet switched network that allows transmission of bursty data.
   Even though TCP constrains a flow's average rate via the window size, it does not control traffic burstiness. In the case of a full, or almost full, queue an arriving burst will cause multiple packet to be dropped. As a result multiple flows may back-off synchronously, and ultimately this could cause a reduction in overall throughput. Since TCP by nature increases its sending rate until packet drops occur, queues will fill up.

> The rationale is that the result of operating a normally-small queue may actually be higher throughput as well as lower end-to-end delay. Queue limits thus should reflect the size of bursts that needs to be absorbed rather than the steady state queues that it is desirable to maintain.
> The problem with the drop-tail queuing discipline in a TCP endpoint environment is that it allows for queues to be full or almost full for sustained periods of time. This is because congestion is signaled to the edges via packet drop only when the queue is already full.

The lock-out problem can be solved by alternative queuing disciplines: "random drop on full" where a randomly selected packet from the queue is dropped at the arrival of a new packet when the queue is full, or "drop front on full" where the packet at the front of the queue is dropped when the queue is full and a new packet arrives. These schemes, however, do not solve the full-queue problem.

Since standard TCP reacts on packet loss, the full-queue problem can be solved by letting routers apply an AQM scheme that drops packets before buffers are full. Clearly the most effective congestion detection can be performed by the router itself. It is only the router that has a unified view of the queuing behavior over time. By dropping packets at the incipient of congestion this information can be communicated to the senders which in response can reduce their load on the network at an early state such that queues can be kept small. Thus the capacity to absorb traffic bursts without dropping packets will be increased. Avoiding queue overflow and the resulting drop of several packets furthermore prevents global synchronization of flows throttling down their rates simultaneously. Maintaining small queues also reduces end-to-end delay seen by flows, this is of importance for delay sensitive traffic such as, e.g., short Web transfers or VoIP. Avoiding full queues and ensuring that there is buffer space available for incoming packets most of the time automatically prevents lock-out behavior. Moreover it prevents router bias against small but highly bursty flows.

Discussion

The traditional approach to the AQM problem is to regard the endpoint algorithms, typically standard TCP, as the process to be controlled, and the link arrival rates and queue sizes as measured outputs. The question is then what control signal that should be chosen and how it should be updated. In the case of standard TCP the control signal thus is packet drop at packet level (or packet drop probability in a macroscopic perspective), and the design problem is how these drops should be generated.

However, considering an environment with delay-based protocols such as TCP Vegas or FAST TCP and interpreting it in an AQM context, the control signal would be delay and the AQM is rather the queuing mechanism itself.

In a general setting any control signal reflecting the amount of congestion could be considered, cf., the explicit feedback protocols discussed in Section 2.2.4. Such schemes takes a unified approach to network congestion control and designs the endpoint protocol rate control and the router mechanism simultaneously.

2.3.2 Overview

We will now review some AQM algorithms proposed to complement the TCP congestion avoidance mechanisms in the endpoints.

TCP friendly AQM algorithms

Already in the early 90s the Random early detection (RED) algorithm (Floyd and Jacobson, 1993) was introduced as an extension of the drop-tail queue management to prevent loss-synchronization and to control queue lengths. This seminal work has been followed by a large number of AQM proposals. Only a few of the most well-known ones are included here.

RED The idea of RED is that the existence of queues is an early sign of network congestion, but that transient congestion and longer-lived congestion must be separated. It is only possible to control longer-lived congestion.

While transient congestion is characterized by a temporary increase in the queue, longer-lived congestion can be observed by an increase in the average queue size. Consequently RED monitors the average queue length, by filtering the queue size samples, and drops packets in a randomized fashion with a probability that is set as an increasing piece-wise linear function of the average queue length. The larger the average queue is, the more congestion and, thus, the higher probability that a packet is dropped.

RED is widely deployed in Internet routers today, it is however, rarely enabled due to that parameters have shown to be hard to tune. One proposed solution to this is to adapt some of the parameters as well (Feng *et al.*, 1997, 1999), the result is called Adaptive RED.

SRED The SRED queue management (Ott *et al.*, 1999), or Scalable RED, explores that a small number of large TCP flows behave differently than a large number of small TCP flows. The number of flows is estimated, and the probability that a packet is dropped is computed based on this estimate and the size of the queue. The flow estimation is accomplished without maintaining a per-flow account and the steady-state average queue is independent of the number of flows.

DRED and PI The DRED algorithm (Aweya *et al.*, 2001), or Dynamic-RED, tries to stabilize the average queue length at a given size without any knowledge about the number of active TCP connections. In principle DRED is identical to RED, they both adjust the packet drop probability as a function of the averaged queue size, what differs between them is the specific drop probability function. DRED takes a control theoretic approach and poses the design of the drop probability function as a reference tracking control problem. By introducing integral action in the control loop, the control objective of a load independent steady-state queue is accomplished for a wide range of load levels.

A closely related approach as in the design of DRED is taken in (Hollot *et al.*, 2001b), the algorithm is referred to as PI after its controller structure (proportional-integral). Also here the deviation from a desirable queue length is suppressed by integral control. However, the controller in (Hollot *et al.*, 2001b) also incorporates a proportional part and thus has one more degree of freedom than in DRED. Furthermore, the parameter tuning is model based.

BLUE A fundamentally different approach, compared to AQM algorithms using queue length as congestion indication, is taken in BLUE (Feng *et al.*, 2002). In BLUE packet loss and link idle events are used to control congestion. The packet drop probability for an enqueued packet is increased if packets are dropped due to buffer overflow. In the case that the queue is empty or the link is idle the probability of drop is decreased. In this manner BLUE continuously searches for an appropriate drop rate.

REM In the REM (Athuraliya *et al.*, 2001), Random exponential marking, queue management scheme an artificial metric, a “price”, is constructed as a measure of congestion. The price is constructed from the weighted combination of the mismatch between the averaged queue length and a reference value, and the difference between the rate of the incoming traffic and the link capacity.

Packets are dropped randomly, and the packet drop probability function is a smooth concave function in the price.

Using this strategy REM tries to stabilize the input rate around the link capacity and the queue around the target independently of the number of flows traversing the link.

AVQ To be able to operate at a minimal queue size the AVQ (Kunniyur and Srikant, 2004), the Adaptive Virtual Queue, algorithm maintains a virtual queue to detect congestion at an early stage. The virtual queue has a (virtual) capacity that is less than the actual capacity of the link. At the arrival of a packet to the physical queue, the virtual queue is updated as well to reflect the new arrival. If the virtual queue overflows at a virtual enqueuing the corresponding real packet in the real queue is dropped.

The virtual capacity is furthermore adapted such that packets are dropped more frequently when the link utilization exceeds a target utilization, and less frequently if link utilization is below the desired value.

An additional feature with AVQ is that instead of a drop-tail policy in the virtual queue, any other AQM with desirable properties can be used on top.

Explicit congestion notification

An alternative to use packet drops to indicate congestion for an AQM scheme is to use the Explicit congestion notification (ECN) specified in (Ramakrishnan *et al.*, 2001). This optional feature requires support at both ends of a connection. When the ECN option is enabled, a router supporting it may instead of dropping a packet mark it by setting bits in the IP header to signal congestion to the endpoints. The sender is then supposed to take the same action as if a packet drop was detected.

The obvious advantage with using ECN is that the number of dropped packets in the network is decreased and consequently also the number of packets that has to be retransmitted. To achieve any significant improvements in terms of user response time by employing AQM, the results in (Le *et al.*, 2007) indicate that ECN must be used in parallel.

2.4 A flow level perspective of Internet congestion control

The complexity of the Internet at *packet level* is tremendous. Thus it is necessary to abstract away irrelevant "details" when analyzing *flow level* performance. A packet level congestion control algorithm imposes specific flow level properties. From a flow level perspective the objective of congestion control is to achieve:

- *Efficient resource utilization.* Links should be utilized in as large extent as possible.
- *Fair resource allocation.* Sharing of resources should be predictable and reasonable.
- *Low queuing delay.* Large delays may hurt delay sensitive applications.
- *Low loss rate.* Retransmitting packets is inefficient.
- *Adaptivity and stability.* The system should adapt to network conditions and have a predictable steady-state operation.

Naturally, the design of a congestion control algorithm can be separated into two levels: the flow level and the packet level. The flow level design aims to achieve certain macroscopic design goals listed above, while the packet level design goal is to implement these flow level goals within the constraints imposed by end-to-end control.

The AIMD scheme discussed in Section 2.2.4 and used by TCP, e.g., is a packet level algorithm with specific flow level properties. Let us as an illustration analyze its steady-state throughput with a simplified example.

Example 2.4.1. *Consider a long-lived TCP flow that is operating in steady-state cycling between incrementing its window with one packet per RTT and halving it when it detects a packet loss. Assume that queuing delay is negligible compared to propagation delay. Consequently the RTT, denoted with τ, is assumed constant. Furthermore assume that the packet loss rate q is 1 packet loss per cycle which occur when the window size is w packets. This implies that the window size varies between $w/2$ and w packets traversing a perfectly periodic sawtooth, see Figure 2.10. If the receiver delivers an ACK for each received packet, the window is increased with 1 packet per RTT and thus each cycle must be $w/2$ RTTs, or $\tau w/2$ seconds The data delivered per cycle is the area under the sawtooth, which is*

$$\left(\frac{w}{2}\right)^2 + \frac{1}{2}\left(\frac{w}{2}\right)^2 = \frac{3}{8}\left(\frac{w}{2}\right)^2$$

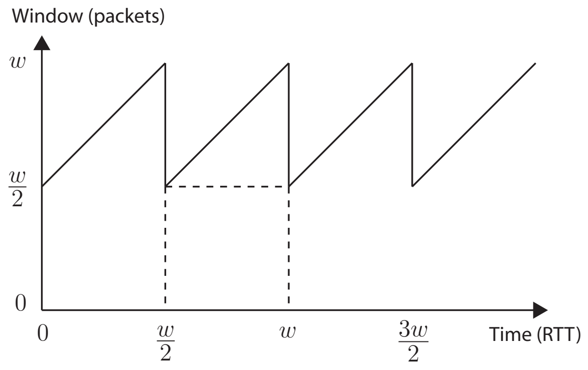


Figure 2.10: TCP window evolution under periodic loss.

packets per cycle. Since 1 packet is lost per cycle the loss rate is

$$q = \frac{8}{3}\left(\frac{2}{w}\right)^2,$$

and by solving for w we get

$$w = \sqrt{\frac{8}{3q}}.$$

The throughput x of the flow is given by the data per cycle divided by the cycle time interval, therefore

$$x = \frac{3w^2/8}{\tau w/2} = \frac{\sqrt{3/2}}{\tau\sqrt{q}} \approx \frac{1.2}{\tau\sqrt{q}} \tag{2.3}$$

packets/s. Early derivations of this famous TCP throughput formula was made by, e.g., Lakshman and Madhow (1997) and Mathis et al. (1997). It was also used without motivation in Section 1.2.1 and mentioned in flight in Section 2.2.4.

We observe form the example that the steady-state throughput of a TCP flow is inversely proportional to its RTT. Thus if two TCP flows are competing about the resources of a bottleneck link, the flow with the smallest delay will achieve a larger part of the capacity. A natural question to ask is: is this really fair?

2.4.1 Fairness

Fairness is a central issue in congestion control. It is of great importance to understand how network resources are allocated among flows/users. Fairness is, however, also a quite ambiguous notion. In a flow perspective, is it fair that each flows achieve the same steady-state throughput? Or is it reasonable, as for TCP, to punish flows with larger delays that are more likely to use more network resources due to its longer path? The answer is not obvious. So, assume we agree in the former definition of per flow fairness, that each flow should have the same throughput. Is it then fair that an application opens up multiple flows to get a larger part of the network capacity? This is unclear as well.

We will here only consider per flow fairness. Let sender i have rate x_i then the rate vector x is a vector with elements x_i. The *rate vector* is said to be *feasible* if each element x_i is non-negative and the total rates at links are within capacity constraints. A rather general definition of fairness that can be found in (Mo and Walrand, 2000) is:

Definition 2.4.1 ((p, α)-Proportionally Fair)**.** *Let $p = (p_1, \ldots, p_N)$ and α be positive numbers. A vector of rates x^* is proportionally fair if it is feasible and for any other feasible vector x*

$$\sum_i p_i \frac{x_i - x_i^*}{{x_i^*}^\alpha} \leq 0.$$

The definition of (p, α)-proportionally fairness covers and formalizes the most common notions of fairness: *max-min fairness* and *proportional fairness.*

Max-min fairness

A max-min fair resource allocation policy maximizes the minimum sending rates. A rate vector x is said to be max-min fair if it feasible and if any individual rate x_i cannot be increased without decreasing some other element x_j which is smaller or equal to x_i. Formally, the condition of a max-min fair resource allocation is achieved by setting weights p_i to unity and letting α approach infinity in the definition of a (p, α)-proportional fair allocation. For unequal weights p_i the allocation is said to be *weighted max-min fair.*

Proportional fairness

A feasible rate vector x^* is proportionally fair if the aggregate of the proportional difference with any other feasible rate vector x is non positive, i.e.,

$$\sum_i \frac{x_i - x_i^*}{x_i^*} \leq 0.$$

This corresponds to setting $p_i = 1$ and $\alpha = 1$ in Definition 2.4.1. For arbitrary weights p_i the resource allocation is called *weighted proportionally fair.*

2.4.2 Discussion

In a historical perspective, the ubiquitous congestion control protocol TCP NewReno and its predecessors TCP Reno and TCP Tahoe have been developed in an evolutionary process based on packet level implementations relying on engineering heuristics. The resulting flow level properties of the protocol, such as fairness, stability, and the relationship between window size and loss probability in steady-state, were not understood until later. As new theoretical tools have become available, which has led to that the understanding of Internet congestion control has increased, the design process is nowadays reversed. Flow level models and objectives now typically guides modern packet level implementations. Recent protocols that could be considered model based to some extent are, e.g., FAST TCP, HighSpeed TCP, Scalable TCP and TCP Illinois.

2.5 The mathematics of Internet congestion control

While an experimental study may provide empirical evidence that a particular implementation fulfills its design goals, it does not explicitly reveal why that is the case. To better understand the mechanisms behind a successful (or unsuccessful) design, mathematical modeling is an essential tool. The objective of the modeling is to reveal flow level properties of the packet level system.

In the area of telecommunications, queueing theory (Kleinrock, 1975) has traditionally been used as a vehicle for analysis and design. In queuing theory the packet is the essential unit and stochastic models are used to characterize packet inter-arrival times, which then are used to derive probability distributions of quantities of interest such as queing delay and packet loss. Due to cumbersome mathematics, achievable results using the machinery of queing theory heavily rely on the abstraction of the arrival process. Typically simplifying assumptions like identically independent distributed inter-arrival times is needed, e.g., an arrival pattern according to a Poisson process gives particularly nice results.

Congestion control mechanisms are based on feedback, unfortunately this introduces correlation in time which typically implies awkward mathematics using the framework of queuing theory. The use of queing theory in Internet congestion control analyses has thus been limited. However, recently significant progress in the theoretical understanding of network congestion control has been made following seminal work by Kelly *et al.* (1998) (see also (Kelly, 1999), the surveys (Kelly, 2003a), (Low and Srikant, 2004), (Chiang *et al.*, 2007) and the book (Srikant, 2004)). The key to success is to work at the correct level of aggregation, namely, fluid flow models ignoring detailed packet level dynamics, and to explicitly model the network congestion measure signal fed back to senders.

2.5.1 The fluid flow approximation

The flow in a digital communication network is discrete, the commodities (bits and/or packets) are transferred in a discrete fashion rather than continuously. More specifically, in a packet switched network, packets are transfered individually in isolated time epochs. Note that this implies that links are in one of the two states busy or idle in practice. The discrete nature of communication networks makes them inherently hard to model in detail and exact analysis is often intractable. It is therefore of relevance to choose the correct level of model aggregation, preferably with the application in mind, to arrive at a model suitable for the type of analysis desired.

In fluid flow modeling the complex packet level "high frequency information" is abstracted away. Packet flows, discrete by nature, are approximated as continuous fluid-like flows and linked by (delay) differential equations.

In queueing theory this involves smoothing of discrete stochastic processes. An arrival process of customers to a queue, e.g., can be smoothed out by averaging over many independent copies of its sample-path. The resulting simplified model can hopefully be used to reveal performance indexes of the originating system. Let us illustrate this with an example.

Example 2.5.1 (Queue dynamics[1])**.** *Assume that we have packets arriving to a queue as a Poisson process with time-varying intensity $x(t)/\rho$, where $x(t)$ is continuous and ρ is the packet size. Also assume that the service times are exponentially distributed with parameter ρ/c, independent of the arrival process. This is an M/M/1 queue with arrival rate $x(t)/\rho$ and service rate c/ρ.*

Let $N(t)$ denote the number of packets in the queue at time t. The amount of queued data is then $b(t) = \rho N(t)$. We now fix some t, with $b(t) > 0$, and examine the change of the queue size during the interval $[t, t+h]$. Let A_h be the number of arriving packets during this interval, then A_h is Poisson distributed with parameter $\int_t^{t+h} x(s)/\rho ds$. Let S_h be the number of departing packets during the same interval. For small h, we can ignore the possibility that the queue becomes empty, and approximate S_h as Poisson distributed with parameter hc/ρ.

The change in the queue size is then

$$b(t+h) - b(t) = \rho\big(N(t+h) - N(t)\big) = \rho A_h - \rho S_h.$$

From this expression we can easily compute the expected value, and, since A_h and S_h are assumed independent, we can also compute the variance:

$$\mathrm{E}[b(t+h) - b(t)] = \rho \mathrm{E} A_h - \rho \mathrm{E} S_h = \int_t^{t+h} x(s) ds - hc,$$

$$\mathrm{Var}[b(t+h) - b(t)] = \rho^2 \mathrm{Var} A_h + \rho^2 \mathrm{Var} S_h = \rho \int_t^{t+h} x(s) ds + hc\rho.$$

If we keep h fixed, and let $\rho \to 0$, it follows that

$$b(t+h) - b(t) \to \int_t^{t+h} x(s) ds - hc$$

in the L^2 sense. If we divide by h and let $h \to 0$, we find that

$$\frac{db(t)}{dt} = x(t) - c.$$

[1]This example is taken from (Möller, 2008).

When $b(t) = 0$, this equation must be extended with a state constraint,

$$\frac{db(t)}{dt} = \begin{cases} x(t) - c & b(t) > 0, \\ \max(0, x(t) - c) & b(t) = 0. \end{cases}$$

This establishes the usual equation for queuing dynamics as the small packet limit of an M/M/1 queue.

In network congestion control analysis, the fluid flow assumption is more ambiguous. In a deterministic setting, fluid flow modeling basically just means approximating flows of packets as continuous fluids, i.e., smoothing the packet inter-arrivals like in the above example. The objective of this higher level of aggregation is mathematically tractable dynamical models with sufficiently good accuracy over (cruder) time scales of interest for the studied application. The window control mechanism in TCP, e.g., is updated on per round trip time basis. Thus we can expect that dynamics on time scales magnitudes finer does not affect the stability of this mechanism and can therefore be ignored. This supports a modeling approach based on fluid flow approximation, cf., the discussion in Section 3.1. Note that the error in the underlying fluid flow approximation diminishes and ultimately disappears as the number of packets in the (fixed) system approaches infinity (i.e., packet size approaches zero and flows become more fluid like). The key in fluid flow modeling is to identify crucial packet level effects and capture them in the continuous time model.

There may be several sources of randomness in a network. Examples are: AQM mechanisms such as RED that drop packets randomly, users connects/disconnects in a random fashion, randomized scheduling in processors, etc.. Here the fluid flow approximation not only refers to the spatial continuous flow assumption, but also involves smoothing over the involved stochastic variables. The prediction accuracy of such averaged models are not only dependent on the packet density in the system but also that the stochastic components are appearing numerously enough. Typically this means that the number of users must be sufficiently large. Let us exemplify this by deriving a dynamical model of the average behavior of the AIMD algorithm TCP is based on.

Example 2.5.2 (AIMD dynamics)**.** *Consider the standard AIMD algorithm that determines the steady-state operation of TCP. Recall that in AIMD the sending rate is regulated through the window size w which is inflated with $1/w$ packets per received ACK and halved when a congestion event is detected.*

Consider the window w as state variable and consider the packet loss probability q as input signal. Let τ denote the round trip time and assume that every packet is acknowledged. Let us also assume that packets are marked instead of dropped by the network. This models a network with ECN enabled.

By definition, one full window of packets is transmitted each RTT. Thus the average sending rate is $x(t) = w(t)/\tau(t)$. Ignoring the effect of signaling delays the sender receives ACKs with this rate as well due to the ACK-clocking. On average a *fraction* $(1-q(t))$ *of these ACKs are positive, each incrementing the window* $w(t)$ *by* $1/w(t)$*, implying that the average window increase is at the rate of*

$$\frac{x(t)(1-q(t))}{w(t)} = \frac{w(t)(1-q(t))}{\tau(t)w(t)} = \frac{1-q(t)}{\tau(t)}.$$

Similarly, the average arrival rate of negative ACKs each halving the window is $x(t)q(t)$*. The window* $w(t)$ *thus on average decreases with rate*

$$\frac{x(t)q(t)w(t)}{2} = \frac{q(t)w^2(t)}{2\tau(t)}.$$

Summing up the window $w(t)$ *evolves on average according to*

$$\dot{w}(t) = \frac{1-q(t)}{\tau(t)} - \frac{q(t)w^2(t)}{2\tau(t)}. \tag{2.4}$$

From the window update law (2.4) we get that the relation between the equilibrium window w *and the equilibrium packet loss probability* q *is*

$$w = \sqrt{\frac{2(1-q)}{q}}.$$

By substituting the equilibrium relation $w = x\tau$ *into this expression we have that the equilibrium rate is given by*

$$x = \frac{1}{\tau}\sqrt{\frac{2(1-q)}{q}} \approx \frac{1.4}{\tau\sqrt{q}}$$

when the packet loss probability q *is small. Comparing this with the TCP throughput formula (2.3) derived in Example 2.4.1. We observe that the two expressions have the same structure by differs slightly in the constant.*

2.5.2 The standard fluid flow congestion control model

We will now review a network fluid flow model introduced by Kelly *et al.* (1998). This model has become standard in flow level modeling of Internet congestion control.

Consider a network consisting of L interconnected links identified with indices $l = 1, \dots, L$ and with capacities c_l. Assume the network is utilized by N flows defined by a sender that transmits data to a receiver over a fixed path on the network and which are indexed with $n = 1, \dots, N$. For model tractability, detailed packet level information is ignored and rates are modeled as flow quantities. Let R denote a routing matrix, which represents which flows that utilizes which links. It is defined by

$$R_{l,n} = \begin{cases} 1 & \text{if link } l \text{ is used by flow } n, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

Before we continue, let us illustrate this with a simple example.

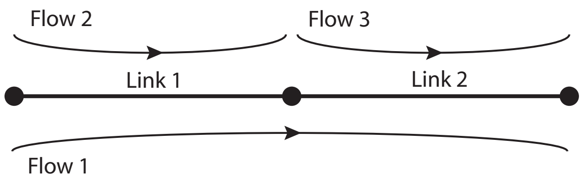


Figure 2.11: Simple network configuration.

Example 2.5.3 (Routing matrix)**.** *Consider the network configuration displayed in Figure 2.11. The* n*th column of the routing matrix gives the path of the* n*th flow, and the* l*th row tells which flows that traverses the* l*th link. It is realized that the routing matrix for this network is*

$$R = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Let $x_n(t)$ be the rate by which sender n transmits data. In this notation the total, or aggregate, rate on the lth link is given by

$$y_l(t) = \sum_{n=1}^{N} R_{l,n} x_n(t) \tag{2.6}$$

if we for simplicity ignore transmission delays. This can be expressed compactly in vector notation, $x = (x_1, \ldots, x_N)^{\mathrm{T}}$ and so on, as

$$y(t) = Rx(t).$$

The key in the modeling is to associate a congestion measure $p_l(t)$ with each link l. We will refer to this positive real valued quantity as *price*. The reason for this naming convention will be explained in Section 2.5.3. The price may represent different physical network quantities. The quantity of the price is specified by the functionality of the protocol that is used. When considering a sender using a loss-based protocol, e.g., the price corresponds to packet drop probability, while in a delay-based setting the price represents the queuing delay at the link. A fundamental assumption of the model is that senders have access to the aggregate price of all links along their path. This means that the signal

$$q_n(t) = \sum_{l=1}^{L} R_{l,n} p_l(t), \tag{2.7}$$

is available for the nth sender. In vector notation we have for all senders

$$q(t) = R^{\mathrm{T}} p(t).$$

As previously the effect of transport delays has been ignored.

To complete the congestion control model we need to specify: how sources adjust their rates based on the aggregate price they observe, and how links set their prices based on their aggregate rates. The former is governed by the endpoint congestion control protocol, like TCP, and the latter by the queue management algorithm at the link, i.e., the AQM. For the discussion here we will postulate a simple first order dynamical law

$$\dot{x}_n = F_n(x_n, q_n) \tag{2.8}$$

to represent the dynamics of the nth source, and similarly

$$\dot{p}_l = G_l(p_l, y_l) \tag{2.9}$$

to model the lth link price update. More general dynamics could of course be considered.

Example 2.5.4. *A delay based congestion control protocol like TCP Vegas or FAST TCP uses queuing delay as congestion measure. The price p_l of a link thus corresponds to the queuing delay of that link, and the aggregated price q_n a sender observes corresponds to the sum of all queuing delays at links along its path. In terms of rates and ignoring static nonlinearities a first-in-first-out (FIFO) scheduled link buffer is simply an integrator integrating the excess rate at the link, i.e., the difference between the aggregate rate y_l and the capacity c_l. The dynamics of the lth link is therefore naturally modeled as*

$$\dot{p}_l(t) = \frac{1}{c_l}\left(y_l(t) - c_l\right) = \frac{1}{c_l}\left(\sum_{n=1}^{N} R_{l,n} x_n(t) - c_l\right).$$

The normalization with the capacity c_l is due to that the price is measured in time rather than buffered packets. To close the loop it remains to specify the rate dynamics of the specific protocol.

The model that explicitly accounts for the feedback in the system is summarized in Figure 2.12. Since introduced by Kelly *et al.* (1998) it has been fundamental for studying equilibrium properties, such as resource allocation, and dynamical properties such as stability and convergence.

2.5.3 Resource allocation

A central issue in a traffic network is to understand how traffic flows between different sources and destinations distribute themselves over the links of the network. It turns out that the key to understand how this is accomplished by decentralized end-to-end control is to pose the network flow control as an optimization problem where the the objective is to maximize the aggregated sender utility (Kelly *et al.*, 1998; Low and Lapsley, 1999).

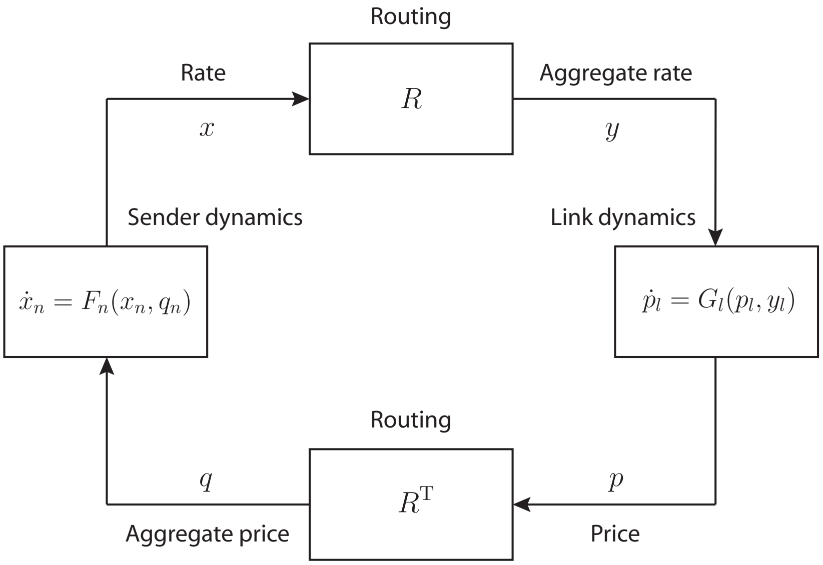


Figure 2.12: The fluid flow congestion control model. Each sender adapts its flow rate dynamically to the amount of congestion in the network. This is represented by the "Sender dynamics" block in the figure and models the endpoint congestion control protocol function. Traffic flows are routed through the network and different flows utilize different links. That is accounted for by the top "Routing" block. The "Link dynamics" block models that links update the congestion measure, the price, as a function of the load on the link. Prices are then fed back to senders, i.e., the lower "Routing" block.

The utility maximization problem

Let us assume that sender n associates a utility $U_n(x_n)$ with a particular sending rate x_n. In a per flow perspective it is probable that each sender increases its utility as the sending rate increases. If utilities are additive, it seems reasonable in a network macroscopic perspective to allocate resources such that the aggregate utility is maximized. Such a resource allocation would thus solve the *utility maximization problem*:

$$\begin{aligned} \max_{x \geq 0} \quad & \sum_{n=1}^{N} U_n(x_n), \\ \text{s.t.} \quad & Rx \leq c, \end{aligned}$$

where vector inequalities are element-wise.

The capacity constraints $Rx \leq c$ and the non-negativity condition $x \geq 0$ defines a convex set with non-empty interior. Thus if the utility functions $U_n(x_n)$ are strictly concave the utility optimization problem is a convex optimization problem that has a unique optimum and can be solved by Lagrangian methods (Boyd and Vandenberghe, 2004).

A centralized solution strategy to the utility maximization problem violates the end-to-end principle. This key idea of the Internet advocates decentralization and implies that control algorithms at endpoints and links should be based on local information. Suppose that we would like endpoint senders to solve the utility maximization problem in a decentralized fashion. The immediate observation is that the rates of the sources are coupled in the shared links through the inequality constraints. This implies that solving for the optimal rate x^* would require cooperation among possibly all sources. Sources thus would have to communicate information about their sending rates to potentially all endpoints in the network, clearly not in line with the end-to-end principle.

Let us formulate the dual utility maximization problem under the assumption that $U_n(x_n)$ are differentiable and concave functions. Introduce the Lagrangian function

$$\begin{aligned} L(x,p) &= \sum_{n=1}^{N} U_n(x_n) - p^{\mathrm{T}}(Rx - c) = \sum_{n=1}^{N} U_n(x_n) - xR^{\mathrm{T}}p + p^{\mathrm{T}}c \\ &= \sum_{n=1}^{N} U_n(x_n) - x^{\mathrm{T}}q + p^{\mathrm{T}}c = \sum_{n=1}^{N} (U_n(x_n) - x_n q_n) + \sum_{l=1}^{L} c_l p_l, \end{aligned}$$

here p is a vector of Lagrange multipliers and $q = R^{\mathrm{T}}p$ its aggregation due to the routing. The dual problem is given by

$$\min_{p\geq 0} \max_{x\geq 0} L(x,p). \tag{2.10}$$

Convex duality implies that an optimum (p^*, x^*) of this dual problem is exactly the solution to the primal utility maximization problem.

An equivalent formulation of the dual problem (2.10) is

$$\min_{p\geq 0} \quad \sum_{n=1}^{N} B_n(q_n) + \sum_{l=1}^{L} p_l c_l \tag{2.11a}$$

where

$$B_n(q_n) = \max_{x_n \geq 0} U_n(x_n) - x_n q_n. \tag{2.11b}$$

The function $B_n(q_n)$ has a nice economic interpretation. Since $U_n(x_n)$ is the utility a source gets when transmitting at rate x_n, $B_n(q_n)$ can be interpreted as a maximization of an individual sender's profit where q_n is the price per unit flow it is hypothetically charged. In this context an individual p_l is interpreted as the price charged per unit flow by link l.

Remarkably, the decomposition of the original utility maximization problem into the sub-problems (2.11a) and (2.11b) allows for a decentralized solution without communication between endpoints. Clearly, if the aggregate price q_n is available to source n it can solve problem (2.11b) without knowledge about the other sources' rates. Thus if the link equilibrium prices p^* can be aligned with the Lagrange multipliers, the optimum that maximizes the individual profits and computed in a decentralized fashion by the senders, will also solve the utility maximization problem due to convex duality. The key to the decentralized implementation is to use link algorithms that depends on local aggregate rates y_l only, and which guarantees that the equilibrium prices p_l^* are indeed Lagrange multipliers. Such an algorithm to be implemented at links is, e.g., the gradient projection algorithm as shown by Low and Lapsley (1999).

Congestion control as a decentralized solution to the utility maximization problem

It turns out that the optimization formulation above provides much insight to the equilibrium properties of network congestion control. Let us study the equilibrium of the system (2.5)–(2.9) defined in Section 2.5.2. Under the assumption of the existence of an equilibrium (p, x), it is characterized by

$$\begin{aligned} F_n(x_n, q_n) &= 0, \qquad q_n = \sum_{l=1}^{L} R_{l,n} p_l, \qquad x_n \geq 0, \qquad n = 1, \ldots, N, \\ G_l(p_l, y_l) &= 0, \qquad y_l = \sum_{n=1}^{N} R_{l,n} x_n, \qquad p_l \geq 0, \qquad l = 1, \ldots, L. \end{aligned}$$

We will assume that the source control equilibrium condition $F_n(x_n, q_n) = 0$ satisfy

$$x_n = f_n(q_n)$$

where $f_n(q_n)$ is a positive, strictly monotone decreasing function. Such an assumption is natural in the context of endpoint congestion control, q_n represents congestion in the source's path and more congestion should imply a smaller equilibrium rate. By assumption the inverse $f_n^{-1}(x_n)$ of the above function exist, i.e., we can write $q_n = f_n^{-1}(x_n)$ in equilibrium. Define the utility function associated with the sender congestion control $F_n(x_n, q_n)$ as the integral of this inverse

$$U_n(x_n) = \int f_n^{-1}(x_n) dx_n.$$

This function is strictly concave by assumption since $U_n'(x_n) = f_n^{-1}(x_n)$. Note that we have by construction that the equilibrium rate coincides with the solution to the convex program

$$\max_{x_n \geq 0} U_n(x_n) - x_n q_n,$$

which exactly corresponds to the profit maximization part of the dual utility maximization problem.

Let us focus on the link algorithms for a while. Assume that the link control equilibrium condition $G_l(p_l, y_l) = 0$ implies that equilibrium rates and prices fulfills

$$\begin{aligned} y_l &\le c_l, \\ p_l(y_l - c_l) &= 0. \end{aligned}$$

While the equilibrium capacity constraint $y_l \le c_l$ is more or less fundamental for any feasible control strategy, the second constraint $p_l(y_l - c_l) = 0$ means that we now are restricted to link algorithms that do not charge a price if it is not congested, i.e., if $y_l < c_l$ in equilibrium then $p_l = 0$. This is a rather mild assumption.

Summarizing the original equilibrium conditions in vector form with the additional constraints we have posed on the system we get

$$\begin{aligned} &x \ge 0, && p \ge 0, \\ &y \le c, && \\ &y = Rx, && q = R^{\mathrm{T}} p, \\ &q_n = f_n^{-1}(x_n) = U_n'(x_n), && n = 1, \ldots, N, \\ &0 = p^{\mathrm{T}}(c - y). && \end{aligned}$$

Remarkably this set of constraints is identical to the Karush-Kuhn-Tucker (KKT) conditions of the utility maximization problem. We know from basic convex optimization theory that when strict convexity and strong duality holds the KKT conditions completely characterize the optimal point. Therefore, under the given assumptions, the equilibrium of the congestion control system also solves the utility maximization problem. Thus the congestion control system can indeed be interpreted as a decentralized solver to this problem. Moreover, the convexity implies that the equilibrium rates x are unique.

We remark that for protocols having a limit cycle behavior in steady-state rather than an equilibrium point, like, e.g., TCP under AIMD, we do not attempt to impose an equilibrium on the detailed dynamics. For those cases we study the average behavior of the protocol. Nevertheless, such an “equilibrium” analysis is useful in understanding the state that is aimed for by the protocol. Illustrations of this appeared in Example 2.4.1 where the “equilibrium” throughput of TCP was derived, and in Example 2.5.2 where the mean window dynamics of the AIMD algorithm was considered.

Utility functions and fairness

We have just seen that under mild assumptions there is a utility function associated with the dynamical law of an endpoint congestion control protocol, and that the protocol utility function defines the equilibrium properties. Consequently, it is convenient to study the fairness properties of a protocol via the utility function.

There is a nice link between (p, α)-proportional fairness, and thus the common notions max-min fairness and proportional fairness, and a simple but quite general family of utility functions which covers several proposed TCP algorithms. It can be shown (Mo and Walrand, 2000) that a class of congestion control protocols with utility functions belonging to the following class of concave functions parameterized by the scalar parameter $\alpha_n \geq 0$ and the positive weight p_n

$$U_n(x_n) = \begin{cases} p_n \log x_n, & \alpha_n = 1, \\ p_n(1-\alpha_n)^{-1} x_n^{1-\alpha_n}, & \alpha_n \neq 1, \end{cases}$$

is (p_n, α_n)-proportionally fair. In particular TCP Vegas, FAST TCP and Scalable TCP have utility functions corresponding to $\alpha_n = 1$. They are thus (weighted) proportionally fair. The utility function of HighSpeed TCP is achieved when $\alpha_n = 1.2$ and for TCP Reno $\alpha_n = 2$. Letting $\alpha_n \to \infty$ corresponds to utility functions of protocols that are max-min fair.

The presented results on fairness hold when considering a homogeneous environment where protocols react to the same type of pricing signal. However, if different protocols based on different pricing share the same network, the equilibrium analysis using the utility maximization framework may not be applicable. In fact it might exist several equilibria as shown in (Tang *et al.*, 2007). For such scenarios the evaluation of fairness becomes formidable.

2.5.4 Dynamics and stability

A predictable steady-state behavior is essential to be able to assess the resource allocation properties of a congestion control system. In the previous section it was implicitly assumed that the control laws at the senders and the links were able to drive the system to a desirable equilibrium—it was assumed that the system was stable.

Stability is important to ensure that fluctuations due to stochastically varying cross traffic are damped, and that the network operates in a favorable region of the state space. Unstable protocols cause small fluctuations in cross traffic to produce large fluctuations in queue lengths, which reduce throughput and increase jitter, which interferes with interactive services such as VoIP.

In a real network the number of users is typically not fixed over time, senders connects and disconnects occasionally. It is desirable that the system adapts to the new equilibrium, imposed by the changed load conditions, sufficiently fast and smooth. Understanding the convergence properties of congestion control algorithms is thus of interest.

Control theory provides a powerful framework for studying dynamical properties of feedback systems like in Figure 2.12. In the aftermath of Kelly's fluid flow formulation, methods of control have successfully been used for analysis of flow level properties of various congestion control systems. Naturally the theory has also been adopted for the purpose of design.

Already in the seminal work by Kelly *et al.* (1998) and Low and Lapsley (1999) stability of the basic schemes was pursued using control theory, however under very idealized settings. Similar work can be found in, e.g., (Kunniyur and Srikant, 2002; Altman *et al.*, 1998; Wen and Arcak, 2004). Results mentioned above have ignored the effect of network delay which is critical for stability. Local stability of TCP Reno with RED when accounting for feedback delays has been studied in (Hollot *et al.*, 2001a; Low *et al.*, 2002b; Tan *et al.*, 2006b). The stability analysis reveals that these protocols tend to become unstable when the delay increases and, more surprisingly, when the capacity increases. This has spurred an intensive research to design protocols that maintain local stability also for networks with high bandwidth-delay product, see e.g., (Floyd, 2003; Kelly, 2003b; Paganini *et al.*, 2005). Other examples of work proving local stability when taking delay into consideration are (Johari and Tan, 2001; Massoulié, 2002; Vinnicombe, 2002).

To be able to achieve global results but still not ignoring delay, the authors in (Deb and Srikant, 2003; Ying *et al.*, 2006; Mazenc and Niculescu, 2003), uses so called Lyapunov-Krasovskii and/or Lyapunov-Razumikhin functionals to establish global convergence. An alternative approach is taken in (Peet and Lall, 2007) where global stability of a TCP/AQM setting is analyzed via integral-quadratic constraints (IQC).

Different control methods suitable for network analysis (focusing on stability issues) are surveyed in (Papachristodoulou *et al.*, 2004).

Results on scalable decentralized stability in networks often exploits special symmetry structures in the way the systems are interconnected. While such assumptions simplifies the mathematical analysis significantly, Lestas and Vinnicombe (2007) find that proofs may break down with an arbitrarily small deviation from protocol symmetry. That is the case for the stability results that appears in (Kelly *et al.*, 1998; Johari and Tan, 2001; Vinnicombe, 2002; Paganini *et al.*, 2005). The observation also highlights the importance of using appropriate models. Note that there may be different dynamic laws with the same equilibrium which are only distinguished by their dynamical properties. Thus a model that is sufficient for analyzing resource allocation may be useless in the sense that it gives incorrect predictions when studying, e.g., stability.

In a recent PhD thesis by Wei (2007) it is pointed out that widely used fluid flow models of (window based) TCP congestion control ignores packet level effects that are crucial for the flow level performance. Analytical predictions based on such models are thus inaccurate which is also confirmed in experiments. It concludes that the ACK-clocking mechanism has a significant impact on TCP performance. Concerning this mechanism models with fundamentally different dynamical properties have been used. In (Altman *et al.*, 2004; Baccelli and Hong, 2002; Hollot *et al.*, 2001a; Liu *et al.*, 2005; Low *et al.*, 2002a) the link is modeled as an integrator, integrating the excess rate on the link resulting in smooth dynamics similar to a first-order filter response. This is contrast to the more recent paper (Wang *et al.*, 2005) which proposes a static model neglecting all transient behavior. A rather comprehensive characterization of the ACK-clocking covering the above mentioned models as special cases will be given in Chapter 3.

2.5.5 Packet level models

A packet switched network such as the Internet is driven by discrete events as the arrival and departure of data packets and protocol time-outs. It is thus naturally modeled as an event-driven system where state transitions occur only at the occurrence of asynchronously generated discrete events. To reduce model complexity and arrive at tractable models to be used for analysis, some simplifying assumptions typically are essential. Considering queuing theory, e.g., this could be assuming a mathematical "well-behaving" packet arrival process.

In the introduction of this section it was mentioned that the use of queuing theory in the presence of feedback many times is formidable. However randomness in network arrivals is a natural assumption when modeling entire connections. Roberts (2001) states that: "Since demand is statistical in nature, performance must be expressed in terms of probabilities and the appropriate modeling tools derive from the theory of stochastic processes", and deduces that like in the design of the telephone network where traffic and queuing theory is fundamental, it should be well suited in the design and analysis of, e.g., admission control or quality of service mechanisms on the Internet where feedback is weaker.

The theory of network calculus can be used to derive deterministic bounds on quantities such as loss and delay from known constraints on traffic and service guarantees, see the survey and (Firoiu *et al.*, 2002) and the book (Le Boudec and Thiran, 2001) for a comprehensive treatment. It may sometimes be preferable to give service bounds with some probability rather than on a deterministic basis. Examples of such work on "stochastic network calculus" can be found in (Kesidis and Konstantopoulos, 2000) and (Chang *et al.*, 2001) where different probabilistic bounds for the same process, a node modeled as a constant rate server, are derived. Like in queuing theory it seems challenging to account for feedback introduced by, e.g., TCP in the framework of network calculus. Typically, quality of service results of networks utilized by TCP relies on simple steady-state throughput or fluid flow models of the feedback mechanism (Sahu *et al.*, 2000; Yeom and Reddy, 2001). However, one exception is the work by Baccelli and Hong (2000) who derives a pure network calculus model of TCP. It is shown that the key features of the protocol can be modeled as a linear dynamical system in the so called max-plus algebra which is the theoretical fundament of network calculus.

While a model may be too complex for analysis it can be suitable for simulation. Event driven network simulation where accurate models of packet networks is implemented in software has shown to be a valuable tool for testing and evaluation. It is also common that data from event driven network simulators is used to benchmark predictions made by simplified models. While this often is appropriate one should not forget that the "true" data is generated by a model, and therefore does not reflect the true system perfectly. In this work we will thus not only rely on data generated using network simulator but also establish results using a physical testbed. Examples of popular network simulators are NS-2, OMNeT++ and OPNET (ns2; omnet; opnet), of which NS-2 is the simulator that has the strongest support in academia.

2.5.6 Hybrid models

While fluid flow models use continuous state variables and packet level models discrete states, hybrid models combine both continuous time dynamics and discrete-time logic. Hybrid network models reduce model complexity by averaging variables that are essentially discrete and approximate them as continuous variables, that could, e.g., correspond to considering the size of a queue that is discrete by nature as continuous. To capture the expressiveness of the original system, continuous states, typically linked by differential equations, are combined with discrete events such as the occurrence of a packet drop and the resulting reaction.

Accurate packet level models are suffering from a large computational burden in large-scale simulations as well as from limited analytical benefits. Fluid flow models overcome these problems by neglecting (hopefully) irrelevant packet level details. Such approximations may sometimes, however, be to coarse. Hybrid models attempts to fill this gap between packet level and fluid flow models.

Currently there is no accurate fluid flow characterization of loss-based TCP flows operating in a network with drop-tail queue management, though this setting reflects the current Internet to a large extent. A framework for hybrid modeling of traffic flows in communication networks is presented in (Bohacek *et al.*, 2003; Lee *et al.*, 2007b). While basic dynamics of queues are modeled as continuous processes, queue overflow and packet drops that characterizes a drop-tail queue are accounted for by discrete events. The framework is used to model a network with TCP and UDP flows and the full model is in large-scale validation experiments shown to be fairly accurate capturing transient phenomena.

Baccelli and Hong (2002) propose a set of fluid evolution equations linked by discrete events to model the AIMD congestion control mechanism used by TCP in a drop-tail network environment. The resulting hybrid model is used in a probabilistic setting to derive steady-state properties such as throughput and fairness presented in the form of autocorrelation functions. A similar model is derived by Shorten *et al.* (2007) who furthermore observes that the model is a positive linear system. This specific structure is explored when characterizing the ensemble-average throughput of the studied network.

2.6 Summary

In this chapter we have provided an introduction to the Internet and the main principles it relies on. We have more thoroughly discussed the Transmission Control Protocol, TCP, and complementary queue management schemes that governs the congestion control on the Internet. Some well-known TCP versions and AQM proposals were also surveyed, and flow level control objectives were discussed. We introduced the network congestion fluid flow model proposed by Kelly *et al.* (1998) which has shown to be extremely valuable for the macroscopic understanding of network congestion control systems. We saw that optimization was a suitable tool to understand equilibrium properties such as fairness and argued that control theory is usable for analyzing dynamical properties of the congestion control system. Some different network models were also briefly outlined.

In the next chapter we will depart from Kelly's standard model and refine it to more accurately model an environment with time-based TCP protocols. In Chapter 4 tools from optimization and control theory will be used for analysis of the model. Results are then confirmed with packet level simulations and testbed experiments in Chapter 5.

Chapter 3

Congestion control modeling

MATHEMATICAL modeling has proven to be an essential tool in the design and analysis of a wide range of engineering systems. However, due to the tremendous complexity of telecommunication systems such as the Internet, the role of mathematical modeling and feedback control theory in the transport layer design process has been modest. The cultural distance between mathematical theory and the desire for simplicity of Internet engineers partially explains this. But more fundamentally has been that, until recently, control theory have had little to offer this radically decentralized, yet highly coupled feedback structure of the system.

In this chapter we present a methodology for deriving dynamical models of packet based networks where congestion control mainly is governed by endpoint window based algorithms such as TCP. The models are suitable for evaluating dynamical properties such as, e.g., stability. The emphasis is on systems of window based schemes that use queuing delay as congestion notification. Many ideas are, however, generic and also applicable to other types of window and rate based algorithms.

3.1 Some general remarks on modeling

In modeling and identification of complex systems, it is instrumental to consider the intended use of the model so that system properties of importance for the application are modeled with sufficient accuracy and irrelevant "details" are ignored. For example, a simulation model may have quite different properties as compared to a model suitable for control design. This is illustrated in the following example.

Example 3.1.1. *The system*

$$G_\circ = \frac{(s+0.05)^2}{(s+1)^2(s+0.01)(s^2+0.01s+0.01^2)}$$

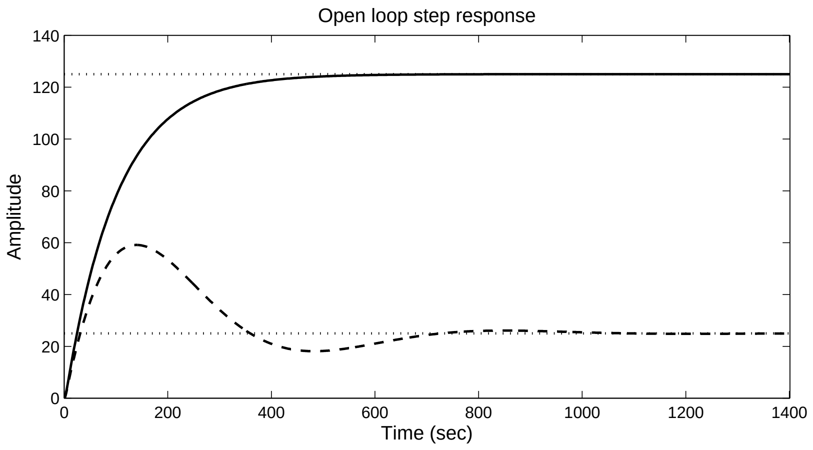


Figure 3.1: Open loop step responses in Example 3.1.1. Dashed line: True system $G_\circ$. Solid line: Model G.

is modeled by

$$G = \frac{0.5}{(s+0.4)(s+0.01)}.$$

The open loop step responses of the system and the model are given in the plot in Figure 3.1. Clearly the model is not suitable for step-response simulation of the open loop system dynamics. This model, however, allows for a reliable control design. Designing a controller

$$C = C(G)$$

such that the designed closed loop transfer function (the complementary sensitivity function)

$$T(G) = \frac{GC(G)}{1+GC(G)}$$

has a bandwidth of 0.4 rad/s and applying this controller to the system yields the closed loop step response in the plot of Figure 3.2, where also the response predicted by the model is shown. We see that the agreement is quite good. □

In order to understand how the controller based on the apparently "bad" model could perform so well we need to consider the control objective. The overall objective of a control design is satisfactory performance of the complementary sensitivity function (the closed loop system)

$$T_\circ(G_\circ, G) = \frac{G_\circ C(G)}{1+G_\circ C(G)},$$

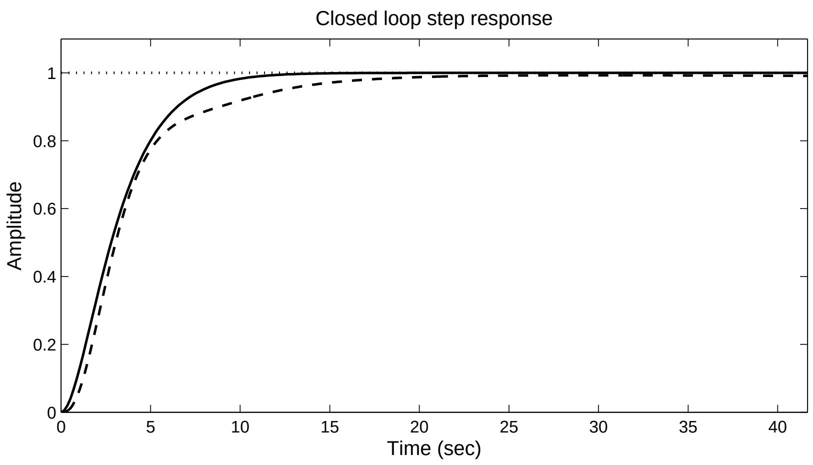


Figure 3.2: Closed loop step responses in Example 3.1.1. Dashed line: True system $G_\circ$. Solid line: Model G.

achieved when the controller $C(G)$ designed on the basis of the model G is applied to the (unknown) true system $G_\circ$. It is thus natural to consider the difference between the nominal design

$$T(G) = \frac{GC(G)}{1 + GC(G)},$$

which has some desirable properties, and the real system $T_\circ(G_\circ, G)$. Various norms of the difference

$$V(G_\circ, G) \triangleq T(G) - T_\circ(G_\circ, G)$$

can be be used to quantify the performance degradation of the true closed loop system compared to the nominal design. Let us consider the $\mathcal{H}_\infty$-norm defined as

$$\|H(j\omega)\|_\infty = \sup_\omega |H(j\omega)|$$

for a scalar transfer function H. In this (induced) norm a characterization of good performance is

$$\|V(G_\circ, G)\|_\infty \ll 1. \tag{3.1}$$

We have that

$$\begin{aligned} V(G_\circ, G) = \frac{G_\circ C(G)}{1 + G_\circ C(G)} - \frac{GC(G)}{1 + GC(G)} &= \frac{(G_\circ - G)\, C(G)}{(1 + GC(G))\,(1 + G_\circ C(G))} \\ &= \frac{G_\circ - G}{G} \frac{GC(G)}{1 + GC(G)} \frac{1}{1 + G_\circ C(G)} = \Delta_G T(G) S(G_\circ, G) \end{aligned}$$

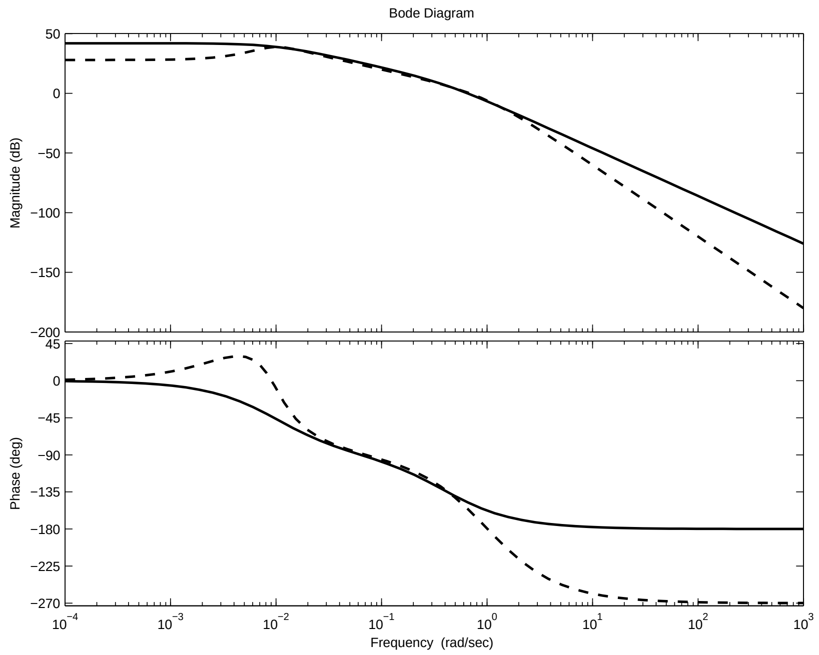


Figure 3.3: Bode diagrams in Example 3.1.1. Dashed line: True system $G_\circ$. Solid line: Model G.

where

$$S(G_\circ, G) = \frac{1}{1 + G_\circ C(G)}$$

is the sensitivity function for the true closed loop system, and

$$\Delta_G = \frac{G_\circ - G}{G}$$

is the relative model error. Thus (3.1) is equivalent to

$$\|\Delta_G \, T(G) \, S(G_\circ, G)\|_\infty \ll 1.$$

Since $T(G)$ typically is approximately unity at low frequencies and rolls off above the desired closed loop bandwidth and $S(G_\circ, G)$ exhibit the opposite behavior it follows that a good relative model fit is only required over a frequency band covering the intended bandwidth of the closed loop system.

Returning to Example 3.1.1 we see from Figure 3.3, which shows the bode plots of the true system $G_\circ$ and the model G, that the model fit is actually good around the desired closed loop bandwidth (0.4 rad/s) for the used model.

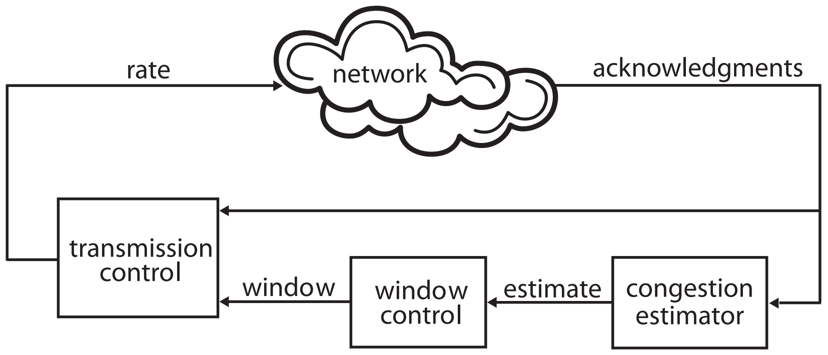


Figure 3.4: Control perspective of window based congestion control.

A prerequisite for the above discussion to be valid is that $S(G_\circ, G)$ is stable which is guaranteed if Δ_G is stable and the closely related robust stability condition

$$\|\Delta_G \, T(G)\|_\infty < 1,$$

is satisfied. The reason why a good model fit is not required at all frequencies is that the closed loop will be insensitive to the model accuracy at low frequencies since the controller gain typically will be large here (this is the essence of feedback control) whereas at high frequencies the controller gain will be small also resulting in low sensitivity to the modeling accuracy.

Packet based network communication systems are highly complex systems exhibiting asynchronous behavior, large number of heterogeneous nodes and nonlinear behavior. The aggregation of traffic flows into fluid flows can be seen as a way of neglecting high frequency behavior well in line with the discussion above. However, as outlined above, for the purpose of, e.g., controller design or stability analysis an appropriate model must also capture the system dynamics around the desired bandwidth. This will be evident when analyzing FAST TCP in Section 4.4.

3.2 Window based congestion control

In window based congestion control, packet delivery reliability is assured through feedback. Coarsely, the receiver acknowledges successfully received packets by sending an ackowledgement packet to the sender. At an ACK arrival the sender decides what information (packet) that is to be (re-)sent and when.

From a dynamical point of view the feedback mechanism can be divided into two separated loops. A block diagram of the control structure in window based transmission control is given in Figure 3.4. The endpoint protocol is at this level of detail represented by the three blocks: transmission control, window control, and congestion estimator. The congestion estimator tries to estimate the level of congestion in the network. This estimate is used by the window control to adapt the window size to an appropriate level. The transmission control regulates the actual sending rate based on this quantity and the rates of the received ACKs.

3.2.1 Congestion estimator

The network state information is indirectly carried from the network to the source (i.e., the sender) by the ACKs. The congestion measure signal must then typically be retrieved by the receiver. In a loss based protocol it has to be concluded if any sent packets did not arrive at the receiver, and subsequently have been dropped. In a delay based scenario, the queuing delay needs to be recreated by subtracting (an estimate of) the propagation delay from measurements of the round trip time that is provided by the ACK. For both cases this is done by the congestion estimator, see Figure 3.4. Here, filtering procedures typically are executed, this to minimize the effect of, e.g., rapid queue fluctuations due to burstiness of traffic.

Consider a system with a fixed number of delay based, “elephants” sources (long lived large flows) sending over a nework with first-in-first-out (FIFO) queueing policy. The network is also utilized by stationary random “mice” traffic (short lived small flows). For simplicity assume that the system is operating in an equilibrium such that queues do not saturate (and hence no packets are dropped). In this “equilibrium” queues are fluctuating around some average value. The fluctuations are due to the stochastic nature of the “mice” cross traffic and packet level effects such as burstiness. The average values of the queue sizes are the “true” equilibrium queing delays that are representative for the amount of congestion in the network. The window based sources, however, will through the received ACKs sample the physical queues and thus the noise polluted “true” queing delays. By low-pass filtering the samples a better estimate of the congestion measure is achieved. Optimal estimation techniques, such as Kalman filtering (Kailath *et al.*, 2000), may also be applied.

As an alternative to indirect signalling, network congestion information could be explicitly signalled from the network to the receiver by utilizing bits in the ACK packet header, cf., Explicit Congestion Notifications (ECN). This, naturally, facilitates the congestion estimation.

3.2.2 Window control

The dynamics of the inner loop in Figure 3.4, the ACK-clocking, is a congestion control mechanism with stabilizing properties in itself. However this feedback control does not provide, e.g., efficient utilization, fairness among flows or responsive adaption to new network conditions. Thus there is in window based congestion control an additional outer control loop which adjusts the window size. Macroscopically, it is the design of this window control that distinguishes different TCPs from each other. The prize of this extra feedback is potential stability problems and the introduction of measurement errors, the drawbacks of feedback control.

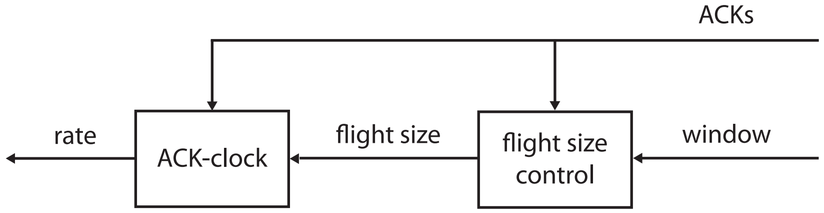


Figure 3.5: Transmission control, split view.

The window control sets the window size based on the congestion level of the network. This is not an explicit network metric. However, it is correlated with measures like packet drops (packet drop probability) or queueing delay which we focus on here. When the network is congested more packets are buffered which implies larger queueing delay, more buffered packets means that buffers operates closer to saturation and concequently packets are dropped more likely. Window control uses such implicit congestion information as control signal.

Protocols like TCP Tahoe and TCP NewReno, use lost packets as congestion indication. While, e.g., TCP Vegas and FAST TCP uses queing delay as primary control signal. Other protocols such as TCP Africa and TCP Illinois use both delay and loss as congestion measures.

3.2.3 Transmission control

The effective rate that the protocol inputs into the network is controlled by the transmission control. A split view of the transmission control is given in Figure 3.5. The transmission of new packets is controlled or “clocked” by the stream of received ACKs. A new packet is transmitted for each received ACK, thereby keeping the number of outstanding packets sent but not yet acknowledged, the flight size, constant. This function is generic for all window based systems and is governed by the ACK-clock. The flight size is adjusted by the flight size control. Typically it is set to match the window size whenever possible. There are, however, physical constraints and potentially traffic shaping algorithms implemented that may need to be accounted for. A negative change in the window size, e.g., can not be realized instantaneously by the flight size control. This would correspond to that packets (or ACKs) inside the network would disappear. The absolute rate of change of any negative change in the flight size is, thus, upper bounded by the received acknowledgment rate. This dependence of the flight size by the ACK rate is indicated in Figure 3.5 through the “ACKs” input. While this fundamental constraint is generic to all window based protocols, protocol specific traffic shaping could also be present. Such shaping typically means smoothing out an abrupt change in the window size over one or a few round trip times to reduce burstiness of the sending rate.

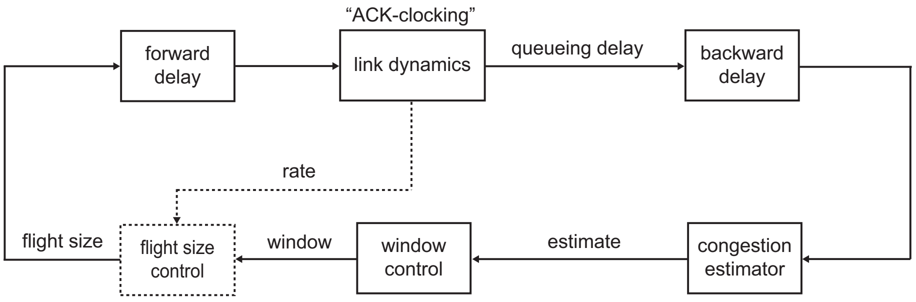


Figure 3.6: Block diagram of the control loop from the perspective of an individual window based congestion control protocol.

3.2.4 Network

A network is a set of interconnected links. Since packets needs to travel through the network they are subject to transport delay. To be able to handle temporary over utilization of a link, network routers operates buffers. A buffer typically integrates the link excess rate and simply drops arriving packets when it reaches its pre-defined point of saturation. A network router may also operate an AQM mechanism dropping packets according to an algorithm or communicating congestion indications to end users by stamping information in packet headers.

Considering delay based protocols only, AQM algorithms will not be discussed here. Furthermore, we will assume that the congestion controllers are able to operate the network sufficiently far from buffer overflow. Thus it is in the sequel assumed that no packet drops occur, and buffer sizes are considered as infinite. We will only model bottleneck links explicitly, any non-bottleneck link is implicitly included in the model as propagation delay. Also, we will not model the ACK path in detail. It is assumed to be uncongested and therefore contributes with propagation delay only.

3.2.5 Closed loop model

When analyzing the flow level performance for a specific window control design (including the estimator) it is convenient to close the inner loop in Figure 3.4. The block diagram shown in Figure 3.6 represents the same system as in Figure 3.4. The "network" (the real network and the ACK-clocking) is in this perspective represented by the blocks: forward delay, link dynamics and backward delay. The two delay blocks represents the propagation delays in a network and the link dynamics contains the ACK-clocking mechanism (which is the interplay between the protocol transmission control and the network). Since window control dynamics often are updated on a RTT time scale, the difference between the flight size and the window size typically is negligible when evaluating the flow level performance of this loop (they are guaranteed to be equal after one RTT). To emphasize this the corresponding feedback is dotted in the block diagram.

The view of window based congestion control taken in Figure 3.6 will be adopted throughout the rest of the thesis. The objective of the remaining part of this chapter is to guide how to, for each block, derive continuous models suitable for evaluating dynamical properties such as stability of the window control loop.

3.3 ACK-clocking

In this part we will derive a model of the dynamics between the protocols flight sizes and the buffer sizes in the network, this is represented by the ACK-clocking block in Figure 3.6.

The key in the modeling is to consider the instantaneous rate a window based sender causes in each queue in the network, and relate that quantity to the flight size.

3.3.1 Preliminaries

A network is modeled as consisting of L links with capacities c_l. Traffic consists of N flows. Let $R = (r_{l,n})$ be the $L \times N$ matrix, with

$$r_{l,n} = \begin{cases} 1 & \text{if link } l \text{ is used by flow } n, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

The matrix R is called the routing matrix. Bidirectional links are modeled as two unidirectional links to account for that traffic flowing in the different directions are separated. Each link maintains a buffer $b_l(t)$, $l = 1, \dots, L$, measured in units of time. The variable $b_l(t)$ thus corresponds to the amount of data in the buffer normalized with the corresponding link capacity and therefore represents the queuing delay. In our delay based context each individual link price $p_l(t)$, $l = 1, \dots, L$, fulfills $p_l(t) = b_l(t)$. Packets are assumed to be transmitted greedily and in FIFO order at links, which reflects the reality of the current Internet.

Let $f_n(t)$ be the number of packets "in flight" (sent but not acknowledged) at time t, i.e., the flight size. The instantaneous rate at which traffic from flow n *enters* link l is $x_{l,n}(t)$, or $x_n(t)$ in the single link case.

The round trip time between the time a packet of flow n enters link l and the time that the "resulting" packet transmitted in response to the acknowledgment of that packet enters link l is denoted $\tau_{l,n}(t)$. It consists of a fixed component d_n due to link propagation delays and a time varying component due to queuing delays. In the single link case, $\tau_n(t) = d_n + b(t)$ where $b(t)$ denotes the queuing delay of the bottleneck link.

Link l may carry cross traffic $x_{l,c}(t)$ which is not window controlled. Cross traffic is assumed for simplicity to not use more than one link in the network.

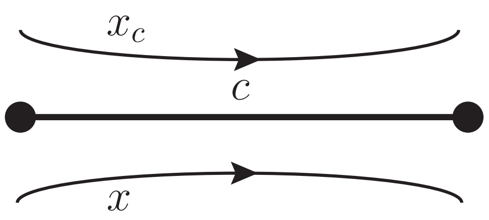


Figure 3.7: Single source single bottleneck configuration.

3.3.2 The single source single bottleneck case

Consider first the simplest case of a single window flow control source sharing a single link with non-window cross traffic of known rate. The configuration is illustrated in Figure 3.7.

In this section, the subscripts will be dropped for clarity, and forward propagation delay is assumed to be zero without loss of generality.

Instantaneous rate

Let us investigate what is known about the instantaneous rate the sender inputs to the queue based on knowledge of the flight size. Note that under the assumption of zero forward propagation delay assumption, this rate is synonymous with the senders transmission rate.

Consider an arbitrary time t. At this time the source has $f(t)$ packets in flight. These packets or their corresponding ACKs could either be located in the buffer or, remember that we are assuming zero forward propagation delay, in propagation between the link and the receiver, or traveling between the receiver and the source. If we for a moment assume that f remains fixed it is clear that the source inputs exactly a flight size f amount of packets into the queue during the interval $(t, t + \tau(t)]$, which is the time it takes for the last packet in the buffer at time t to travel through the queue, to the receiver and the resulting ACK to reach the source again. Note that if the last packet in the buffer at time t is not deriving from the window based source but from the cross traffic, it still holds that the source inputs f packets into the queue during this interval.

Typically $f(t)$ is time varying and thus it may change during the considered interval. What actually matters, however, is the flight size at the end of the interval. Packets transmitted up to time t will be acknowledged by time $t + \tau(t)$. Therefore the number of packets "in flight" at time $t + \tau(t)$, namely $f(t + \tau(t))$, will exactly equal those packets that have been transmitted in the interval $(t, t + \tau(t)]$. This is under the zero forward delay assumption also identical to the number of packets that arrives at the queue during the interval. The instantaneous rate that the window

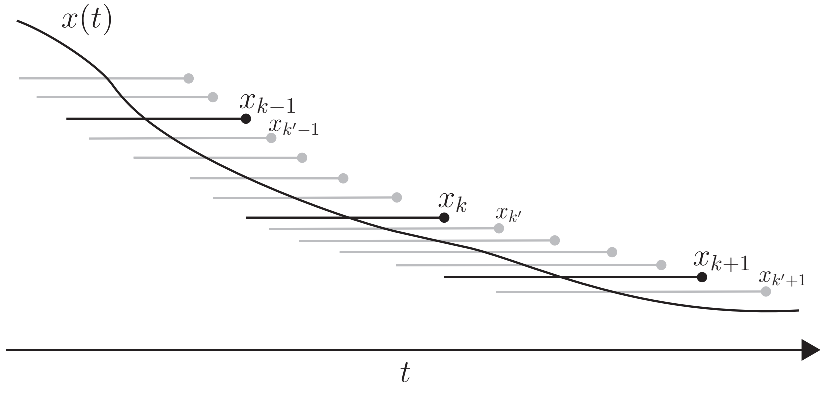


Figure 3.8: The input rate of the source into the queue. The sequences $x_k, x_{k'}, \ldots$ which represents averages over an interval $(t_k, t_k + d + b(t_k)]$, $(t_{k'}, t_{k'} + d + b(t_{k'})]$, $\ldots$ are known. The instantaneous rate is any $x(t)$ with average rate according to the sequences.

based source inputs to the link, $x(t)$, is thus such that the integral equation

$$\int_t^{t+\tau(t)} x(s)ds = f(t+\tau(t)) \tag{3.3}$$

is fulfilled. We emphasize that this holds for all times t. We observe that $x(t)$ is any non-negative function with average value $f(t+\tau(t))$ over the interval $(t, t+\tau(t)]$. Figure 3.8 shows how (3.3) can be interpreted as a sliding window of such averages.

ACK-clocking model

In terms of rates, a link buffer is simply an integrator, integrating the excess rate at the link (modulus static non-linearities present in the system, cf., non-negative constraints/drop-tail queues). Thus, having defined the instantaneous rate $x(t)$, the buffer dynamics is naturally given by

$$\dot{b}(t) = \frac{1}{c}\left(x(t) + x_c(t) - c\right). \tag{3.4}$$

The normalization with the capacity c is due to that the buffer is measured in units of time. The whole system is now described by (3.3)–(3.4). This is a delay *Differential Algebraic Equation* (DAE) which can be expressed in different ways.

Differentiating (3.3) with respect to t gives

$$(1+\dot{\tau}(t))x(t+\tau(t)) - x(t) = (1+\dot{\tau}(t))\dot{f}(t+\tau(t)). \tag{3.5}$$

Rearranging this expression and shifting the time point according to

$$t = \tilde{t} + \tau(\tilde{t}) = \tilde{t} + d + b(\tilde{t})$$

gives

$$\begin{aligned} x(t) = \frac{x(t-\tau(\tilde{t}))}{1+\dot{\tau}(t-\tau(\tilde{t}))} + \dot{f}(t) &= \frac{x(t-\tau(\tilde{t}))}{1+\dot{b}(t-\tau(\tilde{t}))} + \dot{f}(t) \\ &= \frac{cx(t-\tau(\tilde{t}))}{x(t-\tau(\tilde{t}))+x_c(t-\tau(\tilde{t})))} + \dot{f}(t) \end{aligned}$$

Note that the rate at time t is not determined solely by the window and RTT at the current time, but depends on the rate one RTT previously as well.

Assume for clarity constant cross traffic x_c. If we solve for $x(t)$ in (3.4) and plug it in to the integral equation (3.3), the dependence on the instantaneous rate can be eliminated:

$$\begin{aligned} f(t+\tau(t)) &= \int_t^{t+\tau(t)} x(s)ds = \int_t^{t+\tau(t)} \big(c\dot{b}(s) - (x_c - c)\big)\, ds \\ &= cb(t+\tau(t)) - cb(t) - \tau(t)(x_c - c) = cb(t+\tau(t)) - x_c b(t) - d(x_c - c). \end{aligned}$$

This gives

$$b(t+\tau(t)) = \frac{x_c}{c} b(t) + \frac{1}{c} f(t+\tau(t)) + d\frac{x_c - c}{c}, \tag{3.6}$$

that explicitly states how the queue depends on its state one RTT previously and the current flight size $f(t)$. Note that this is a pure delay equation since there are no derivatives involved. It turns out that it is instructive to study this system as a discrete time system.

The continuous time delay system (3.6) can equivalently be expressed at sample points t_k with the discrete time dynamical system,

$$b_{k+1} = \frac{x_c}{c} b_k + \frac{1}{c} f_{k+1} + d\frac{x_c - c}{c}, \tag{3.7a}$$

$$t_{k+1} = t_k + \tau_k = t_k + d + b_k, \tag{3.7b}$$

under the convention $b_k = b(t_k)$. This is a linear but non-uniformly sampled system with sample time $T_k = d + b_k$, which can be analyzed by the means of linear systems theory. The immediate observation is that for the case of no cross traffic, $x_c = 0$, we have

$$b_{k+1} = \frac{1}{c} f_{k+1} - d,$$

and thus a static update from the flight size f to the buffer b for such a scenario. By noting that (3.7a) is a first-order linear filter with a pole in x_c/c we can also

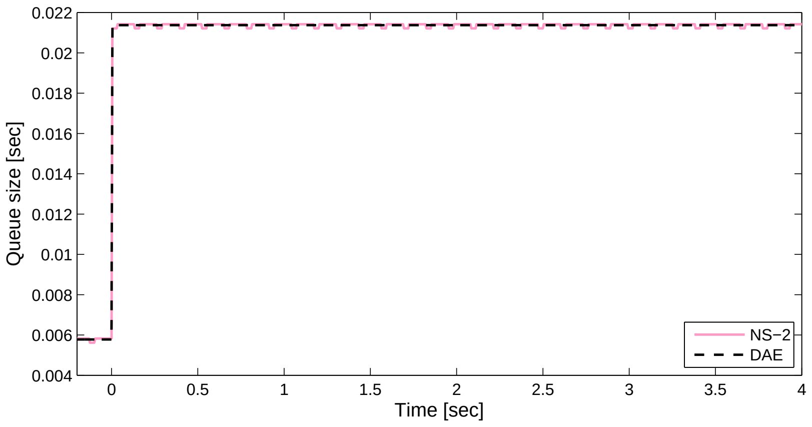


Figure 3.9: Validation experiment. Solid line: NS-2 simulation. Dashed black line: The model (3.3)-(3.4).

conclude from linear systems theory that: the system is stable since $0 \leq x_c/c < 1$, the system will be more transient with increasing amount of cross traffic x_c since the pole then approaches 1, and there will be no oscillations or over-shoot in the case of a step in the flight size f. Let us illustrate our findings with an example.

Example 3.3.1. *Consider a window based source utilizing a single bottleneck link of capacity* $c = 40$ *Mbit/s. The window based source has a round trip propagation delay of* $d = 150$ *ms and its flight size is initially set to a constant size* $f_0 = 750$ *packets, packet size is set to* $\rho = 1040$ *bytes. At time* $t = 0$ *s the system is perturbed from equilibrium by increasing the flight size of the window based source with 75 packets to* $f(t) = 825$ *packets. The solid line in Figure 3.9 is the queue size when this scenario is simulated in NS-2. The dashed black line corresponds to the model (3.3)-(3.4). We observe an immediate increase in the queue, accurately predicted by the model.*

Now, consider a similar experiment but with bottleneck capacity $c = 200$ *Mbit/s. In addition, for this case the link is utilized by (UDP) cross traffic with constant rate* $x_c = 160$ *Mbit/s. The solid line in Figure 3.10 is the queue size when this scenario is simulated in NS-2. The dashed black line corresponds to the model (3.3)-(3.4). For this case we observe a significant transient. This is also accurately predicted by the model.*

Before generalizing the model let us summarize with some key points:

- In terms of rates a queue is just an integrator, this is modeled by (3.4).
- The key is to define the instantaneous rate of which data from the window based source flows into the queue, see (3.3).

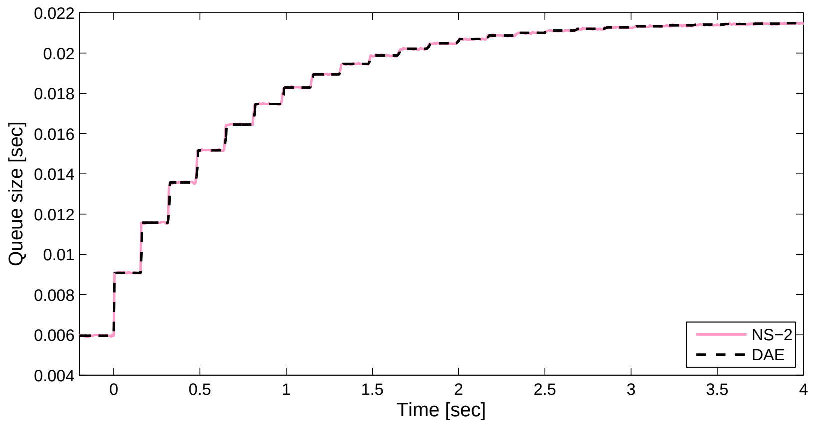


Figure 3.10: Validation experiment. Solid line: NS-2 simulation. Dashed black line: The model (3.3)-(3.4).

- To understand the properties of the system and the effect of non-window based cross traffic, it is instructive to consider the discrete time model (3.7).

Buffer non-negativity constraint

A buffer is non-negative by definition. This needs to be taken into consideration when the system is operating in a nonlinear regime close to this lower saturation. It is straightforwardly accounted for by extending the model (3.3) and (3.4) by adding a non-negativity constraint to (3.4):

$$\dot{b}(t) = \begin{cases} \frac{1}{c}\left(x(t) + x_c(t) - c\right), & \text{if } b(t) > 0 \text{ or } x(t) + x_c(t) - c > 0, \\ 0 & \text{otherwise.} \end{cases}$$

3.3.3 The multiple sources single bottleneck case

When N flows share a single bottleneck link, there are N constraints analogous to (3.3) expressing the rate $x_n(t)$ for each flow. In addition the buffer dynamics can be expressed similar to (3.4). Without loss of generality, assume zero forward propagation delay and this leads to the model

$$\dot{b}(t) - \frac{1}{c}\left(\sum_{n=1}^{N} x_n(t) + x_c(t) - c\right) = 0, \tag{3.8a}$$

$$\int_t^{t+\tau_n(t)} x_n(s)\,ds - f_n(t + \tau_n(t)) = 0, \tag{3.8b}$$

for $n = 1, \ldots, N$.

Like for the single link case this DAE model is easily extended to incorporate present static non-linearities, e.g., the buffer non-negativity constraint.

We will exam how this model performs in Chapter 5.

3.3.4 Accumulated data formulation

It turns out that it sometimes is convenient to express the dynamics in terms of accumulated data instead of rates, e.g., when solving the equations numerically. Let $y_n(t)$ denote the total amount of data that has arrived to the queue up to time t and which derives from the nth source. Then by definition

$$\dot{y}_n(t) = x_n(t), \quad n = 1, \dots, N.$$

Similarly, let $y_c(t)$ denote the total amount of cross traffic data that has arrived to the queue up to time t, so

$$\dot{y}_c(t) = x_c(t).$$

Now we can express (3.8a) as

$$\dot{b}(t) - \frac{1}{c}\left(\sum_{n=1}^{N} \dot{y}_n(t) + \dot{y}_c(t) - c\right) = 0.$$

Integrating both sides, starting at time $t = 0$, gives

$$b(t) = b(0) + \frac{1}{c}\left(\sum_{n=1}^{N} (y_n(t) - y_n(0)) + y_c(t) - y_c(0) - tc\right).$$

Furthermore, from (3.8b) we have, for $n = 1, \dots, N$,

$$\int_t^{t+\tau_n(t)} \dot{y}_n(s)\, ds - f_n(t + \tau_n(t)) = y_n(t + \tau_n(t)) - y_n(t) - f_n(t + \tau_n(t)) = 0.$$

After shifting time point the dynamics thus is given by

$$b(t) = b(0) + \frac{1}{c}\sum_{n=1}^{N} (y_n(t) - y_n(0)) + \frac{1}{c}(y_c(t) - y_c(0)) - t, \tag{3.9a}$$

$$y_n(t) = y_n(t - \tau_n(\tilde{t}_n)) + f_n(t), \quad n = 1, \dots, N. \tag{3.9b}$$

where $\tilde{t}_n$ solves $t = \tilde{t}_n + \tau_n(\tilde{t}_n) = \tilde{t}_n + d_n + b(\tilde{t}_n)$. Notice the similar structure of the recursive update in the accumulated rate in (3.9b) and the queue update in the single flow case (3.6).

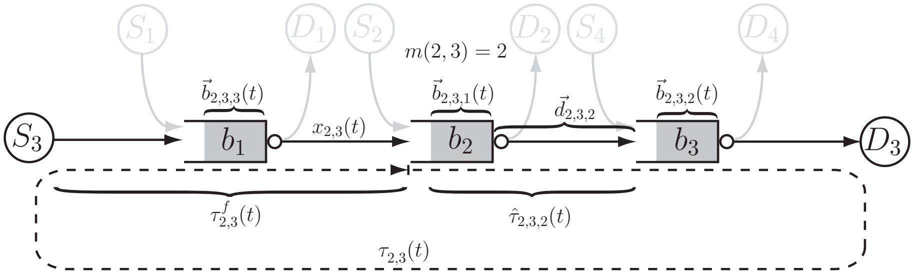


Figure 3.11: Parking lot topology example with 3 bottleneck links and 4 sources. Source i is denoted S_i and its destination D_i. Delays and rates that are shown is for Path 3 in the point of view of Link 2.

3.3.5 The multiple sources multiple bottlenecks case

We will now consider a general network with multiple bottleneck links utilized by multiple sources.

Like for the single link case each buffer integrates the excess rate traversing the link. Recall the definition of the routing matrix R, see (3.2). It is clear that the multi link analogous to (3.8a) is

$$c_l\dot{b}_l(t) = \sum_{n=1}^{N} r_{l,n}x_{l,n}(t) + x_{c;l}(t) - c_l, \tag{3.10a}$$

for all $l = 1, \dots, L$. This expression expresses how the rate of change of the buffer of the lth link is proportional to the total traffic the link carries. It remains to determine $x_{l,n}(t)$ analogously to (3.3). This case is significantly more complex since packets experience delays at different instants of time at each link.

Let $\tau_{l,n}(t)$ be the round trip time from when a packet from source n arrives at link l to the arrival at link l of the "resulting" packet—the packet sent as a result of the acknowledgment of the first. We refer to Figure 3.11 for a graphical illustration of this in a simple "parking lot" topology. Similarly, let $\tau^f_{l,n}(t)$ be the time from when a packet released from source n reaches link l. Let $f_{l,n}(t)$ represent the flight size of source n as if the source would be located directly at link l, then it holds that

$$f_{l,n}(t + \tau^f_{l,n}(t)) = f_n(t). \tag{3.10b}$$

The instantaneous rate $x_{l,n}(t)$ then satisfies

$$\int_t^{t+\tau_{l,n}(t)} x_{l,n}(s)ds = f_{l,n}(t + \tau_{l,n}(t)). \tag{3.10c}$$

It remains to calculate $\tau_{l,n}(t)$ and $\tau^f_{l,n}(t)$. To do this, it is necessary to keep track of the order of the links along each source's path. Let $\vec{b}_{l,n}(t)$ be a column vector of the same dimension as the number of links in the nth source's path, say L_n. The elements of $\vec{b}_{l,n}(t)$ are the buffer sizes in the path of source n, ordered such that the first element corresponds to the buffer size of link l, the second element the buffer size of the link *downstream* of link l in the nth source's path, and so on, and finally the last element corresponds to the link buffer *upstream* of link l. The ith element in a vector $\vec{b}_{l,n}(t)$ is denoted $\vec{b}_{l,n,i}(t)$.

The ordered propagation delay $\vec{d}_{l,n}$ can be defined similarly as for the queuing delays. So $\vec{d}_{l,n,i}$ represents the propagation delay between link l and the link $i-1$ hops after l on path n, and where by convention $\vec{d}_{l,n,L_n+1} = d_n$. Note that $\vec{d}_{l,n,1} = 0$, and, if link l is the kth link and link l' is the k'th link on path n where $k' > k$ (link l' is downstream link l), then $\vec{d}_{l,n,1+k'-k} + \vec{d}_{l',n,L_n+1-(k'-k)} = d_n$ by definition of d_n.

Denote with $m(l,n)$ the order which link l has on path n. Let $\hat{\tau}_{l,n,i}(t)$ be the delay such that a packet which arrives at link l at time t arrives at the link $i-1$ hops after l on path n at time $t + \hat{\tau}_{l,n,i}(t)$. (Strictly, the packet which arrives may be an acknowledgment or a "resulting" packet. Recall that when a packet arrives at the receiver an ACK packet is generated, which when received by the source triggers the transmission of a "resulting" packet.) The total delay, including the queuing at each link, is then

$$\hat{\tau}_{l,n,i}(t) \triangleq \vec{d}_{l,n,i} + \sum_{k=1}^{i-1} \vec{b}_{l,n,k}(t + \hat{\tau}_{l,n,k}(t)). \tag{3.10d}$$

It follows that the interval of integration in (3.10c) is

$$\tau_{l,n}(t) = \hat{\tau}_{l,n,L_n+1}(t) = d_n + \sum_{i=1}^{L_n} \vec{b}_{l,n,i}(t + \hat{\tau}_{l,n,i}(t)). \tag{3.10e}$$

Similarly, the forward delay linking $f_{l,n}(t)$ with $f_n(t)$ is

$$\tau^f_{l,n}(t) = \hat{\tau}_{\ell(n),n,m(l,n)+1}(t), \tag{3.10f}$$

where $\ell(n)$ is a "link" located at the source of flow n, introduced to model propagation delay between the source and the first (bottleneck) link included in the routing.

Summarizing, the model of the ACK-clocking dynamics for a system of N window based sources utilizing a network of L links is given by (3.10). If necessary the model is easily extended with non-negativity constraints like for the single flow single link case, see Section 3.3.2. The accuracy of the model is investigated in Section 5.2.2.

3.4 Protocol dynamics

It remains to derive fluid models of the lower blocks of Figure 3.6: the window control, the congestion estimator and the flight size control, all together referred to

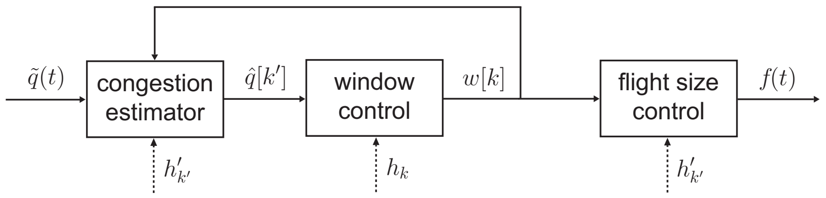


Figure 3.12: Endpoint control structure.

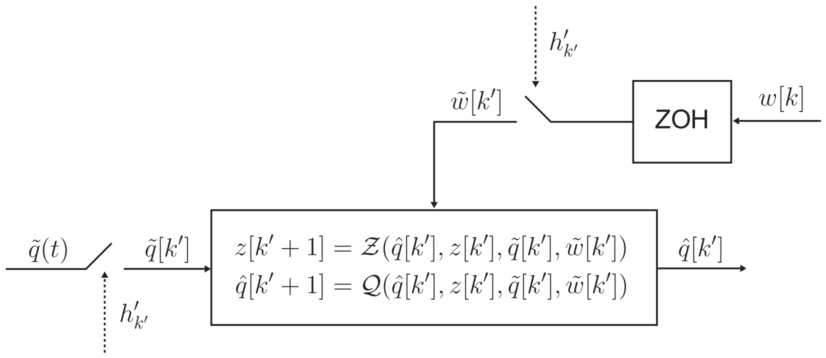

$h'_{k'}$
$\tilde{w}[k']$
ZOH
$w[k]$
$\tilde{q}(t)$
$\tilde{q}[k']$
$z[k'+1] = \mathcal{Z}(\hat{q}[k'], z[k'], \tilde{q}[k'], \tilde{w}[k'])$
$\hat{q}[k'+1] = \mathcal{Q}(\hat{q}[k'], z[k'], \tilde{q}[k'], \tilde{w}[k'])$
$\hat{q}[k']$
$h'_{k'}$

Figure 3.13: Split view of the protocol estimator procedure.

as the protocol dynamics. This is the theme of this section.

The structure of the window control mechanism may differ between protocols. It is thus not possible to present generic models as we did for the ACK-clocking in Section 3.3. However, the objective is to provide guidelines for how to derive fluid flow models of the window based source.

A quite general picture of the dynamics of a congestion control protocol is given in Figure 3.12. The by noise affected congestion measure $\tilde{q}(t)$ is carried to the source with the returning ACKs. An estimate $\hat{q}[k]$ is produced which is used as input to the the window control. The window size $w[k]$ is used as a reference value for the flight size and it may also be used by the estimator. The system is typically event driven, and the different parts of the algorithms typically operate at different (non-uniform) sampling rates, in Figure 3.12 denoted by $h'_{k'}$ and h_k. The ACK inter-arrival time and the round trip time are natural choices of sampling times, but they can of course be chosen arbitrarily. Next, the different parts of the system are discussed in more detail.

3.4.1 Estimator dynamics

A detailed view of the estimator is given in Figure 3.13. The sampling is typically event based and a sample of the network congestion state is collected whenever an ACK arrives. Note, however, that the absence of arriving ACKs in fact provides bounds on the minimum queuing delay and thus network state information is accessible any time. If we, e.g., assume sampling at the arrival of an ACK, the sampling time $h'_{k'}$ corresponds to the ACK inter-arrival time. A sampler with sampling instants t_k is mathematically described by

$$g[k] = \int_0^\infty \delta(s - t_k) g(s)\, ds, \qquad k = 0, 1, 2, \dots, \tag{3.11}$$

where $\delta(\cdot)$ is a Dirac impulse. Note that prior to the sampling anti-alias filtering should occur, this is however often neglected in network congestion control. The price sample $\tilde{q}[k']$ is driving the estimation algorithm that according to some update law produces a price estimate $\hat{q}[k']$ that is fed to the window control algorithm. Let z be a state vector and let us postulate the following dynamic update law for the estimator,

$$\begin{aligned} z[k'+1] &= \mathcal{Z}(\hat{q}[k'], z[k'], \tilde{q}[k'], \tilde{w}[k']), \\ \hat{q}[k'+1] &= \mathcal{Q}(\hat{q}[k'], z[k'], \tilde{q}[k'], \tilde{w}[k']). \end{aligned}$$

The estimator update mechanism may also use internal protocol state information such as, e.g., the window size w, this has been assumed in Figure 3.13. The window control typically samples on a round trip time basis. In that case the window size information needs to be hold and re-sampled at the rate the estimator operates at. This is done by the ZOH block together with the adjacent sampler in Figure 3.13. The ZOH block represents a zero-order hold function defined by

$$g(t) = g(t_k), \qquad t_k \le t < t_{k+1}. \tag{3.12}$$

Other types of hold functions could, of course, also be considered, cf., a first-order hold (Åström and Wittenmark, 1997).

While protocols operate in a discrete time fashion, we would for analytical purposes like to have continuous time models that fit into the fluid flow modeling framework. We thus seek the continuous functions $Z(\hat{q}, z, u_q, u_w)$ and $Q(\hat{q}, z, u_q, u_w)$, such that the system in Figure 3.14 has the same input output behavior at sampling instants as the original system in Figure 3.13.

As an illustration, consider a simple family of estimators with the following structure

$$\hat{q}[k'+1] = \mathcal{Q}_1(q[k'], w[k'])\hat{q}[k'] + \mathcal{Q}_0(q[k'], \tilde{w}[k']). \tag{3.13}$$

Based on this structure it seems reasonable to assign

$$\dot{\hat{q}}(t) = Q(\hat{q}, u_q, u_w) = Q_1(u_q, u_w)\hat{q}(t) + Q_0(u_q, u_w), \tag{3.14}$$

where Q_0 and Q_1 are to be detected so that (3.14) coincides with (3.13) at the sampling instants. The inputs $u_q(t)$ and $u_w(t)$ are constant over the sampling

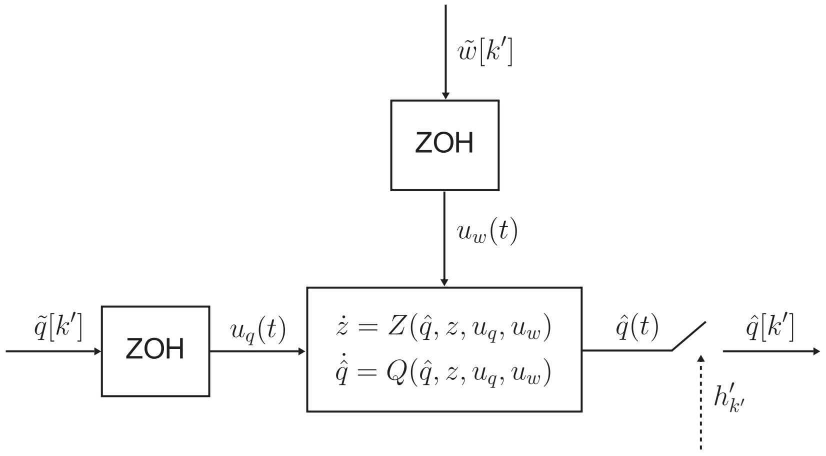


Figure 3.14: Continuous time equivalent estimator dynamics.

interval $t_{k'} \leq t < t_{k'+1}$, which implies that $Q_1(u_q, u_w)$ and $Q_0(u_q, u_w)$ are as well. Solving for the state at t_{k+1} thus gives

$$\begin{aligned}\hat{q}(t_{k'+1}) &= \exp(Q_1(u_q(t_{k'}), u_w(t_{k'}))h'_{k'})\hat{q}(t_{k'}) \\ &\quad + Q_0(u_q(t_{k'}), u_w(t_{k'})) \int_0^{h'_{k'}} \exp(Q_1(u_q(t_{k'}), u_w(t_{k'}))s)ds, \quad (3.15)\end{aligned}$$

where $h'_{k'} = t_{k'+1} - t_{k'}$ is the sampling interval of the estimator (which typically corresponds to the ACK inter-arrival time). The unknowns Q_1 and Q_0 are now possible to identify from (3.13) and (3.15), we have

$$\begin{aligned}\mathcal{Q}_1(u_q, u_w) &= \exp(Q_1(u_q, u_w)h'_{k'}), \\ \mathcal{Q}_0(u_q, u_w) &= Q_0(u_q, u_w) \int_0^{h'_{k'}} \exp(Q_1(u_q, u_w)s)ds,\end{aligned}$$

and thus

$$Q_1(u_q, u_w) = \frac{\log(\mathcal{Q}_1(u_q, u_w))}{h'_{k'}}, \quad (3.16)$$

$$Q_0(u_q, u_w) = \frac{\mathcal{Q}_0(u_q, u_w)\log(\mathcal{Q}_1(u_q, u_w))}{h'_{k'}\,(\mathcal{Q}_1(u_q, u_w) - 1)}. \quad (3.17)$$

Note that the existence and uniqueness of (3.16) and (3.17) is dependent on the the original discrete update (3.13) and subsequently $\mathcal{Q}_0$ and $\mathcal{Q}_1$. Consider for example a stable linear update with a real negative pole, i.e., Q_1 is a constant and $-1 < \mathcal{Q}_\infty < 0$. For this case there are many different continuous models that are equal at sampling instants.

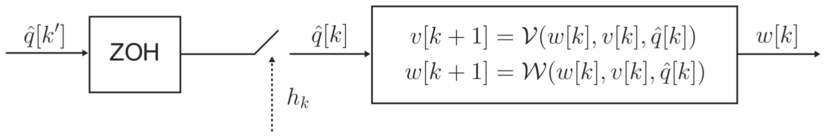

$\hat{q}[k']$
ZOH
$\hat{q}[k]$
h_k
$v[k+1] = \mathcal{V}(w[k], v[k], \hat{q}[k])$
$w[k+1] = \mathcal{W}(w[k], v[k], \hat{q}[k])$
$w[k]$

Figure 3.15: Split view of the window control.

The continuous time model is exact at sampling instants and therefore of similar complexity as the original discrete system. However, by appropriate simplifications of the sample interval $h'_{k'}$ and the zero-order hold functions in the continuous time model, we can derive models that meets the accuracy the application requires and which, hopefully, are more tractable.

Consider a linear stability analysis of a protocol around an equilibrium point. Assume that the network is sampled at every ACK arrival and where the window control is low-pass. Over one RTT a window amount of ACKs arrives at the source. The average ACK inter-arrival time, and sample interval, thus equals the equilibrium round trip time divided with the equilibrium window size. This seems like a suitable sample interval approximation for the application. The low-pass property of the window control implies that the model should be quite robust to the high frequency model errors introduced when ignoring the sample time variance.

3.4.2 Window control

A split view of a generic window update mechanism is given in Figure 3.15. Network congestion state estimates $\hat{q}[k']$ are received from the congestion estimator. The information is then often down sampled to a slower rate h_k. This is governed by the zero-order hold block and the sampler in Figure 3.15, recall (3.11) and (3.12), and should be proceeded with an anti-aliasing filtering operation. Typically the window control works on a round trip time basis while the estimator samples and operates per ACK arrival. The re-sampled congestion estimate $\hat{q}[k]$ is used as input in the window size update law which we pose as

$$\begin{aligned} v[k+1] &= \mathcal{V}(w[k], v[k], \hat{q}[k]), \\ w[k+1] &= \mathcal{W}(w[k], v[k], \hat{q}[k]), \end{aligned}$$

with possible internal states collected in the vector v, and subsequently the window size $w[k]$ is fed to the flight size control.

Like for the estimator case we seek a continuous time equivalent control counterpart to the purely discrete update law in Figure 3.15. In other words we seek the continuous functions $V(w, v, u_{\hat{q}})$ and $W(w, v, u_{\hat{q}})$ such that for identical inputs the output of the system in Figure 3.16 and the output of the original system in Figure 3.15 is equivalent at sampling instants. To complete this we need additional information about the system and we refer to Section 3.4.1 for an illustration of

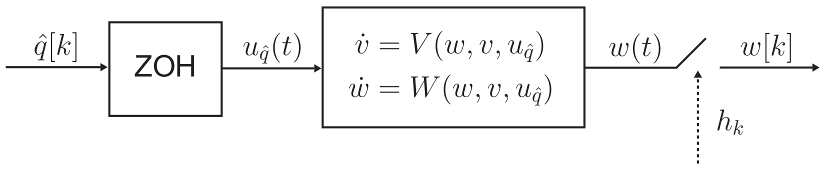

$\hat{q}[k]$
ZOH
$u_{\hat{q}}(t)$
$\dot{v} = V(w, v, u_{\hat{q}})$
$\dot{w} = W(w, v, u_{\hat{q}})$
$w(t)$
$w[k]$
h_k

Figure 3.16: Continuous time equivalent window control.

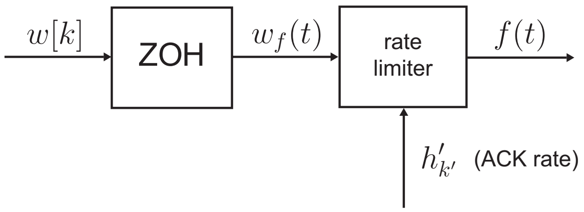


Figure 3.17: Flight size control split view.

how this can be done for a simple system. The final step is then to simplify the model, i.e., to find suitable approximations of the zero-order hold, the non-uniform sampling times h_k, and the continuous time equivalent window control. This should be done with the model application in mind such that the accuracy and complexity of the model obtained matches the intended use.

3.4.3 Flight size control

Neglecting any smooth realization of changes in the window, the flight size control objective is to track the window size, i.e., to keep the difference between the number of unacknowledged packets and window size as small as possible. For most scenarios it seems reasonable to assume that window increments can be followed perfectly. When the window size is increased with Δw packets the flight size control just inputs Δw packets into the network without waiting for any ACKs. A negative change in the flight size, however, correspond to receiving an ACK without sending a new packet. Thus the rate that the flight size can be decreased with cannot exceed the rate of the returning ACKs. Note that by construction, the flight size is guaranteed to be equal to the window size within one RTT. The difference between the window size and the flight size is thus in a sub-RTT time scale.

A model of the flight size control is shown in Figure 3.17. The reference window size $w[k]$ is received from the window control. It is held constant during the window control sampling intervals. The held signal is denoted $w_f(t)$. Together with the ACK rate signal it is fed into the rate limiter block. This block models the constraints on the rate of change of the flight size $f(t)$.

A block diagram of a rate limiter in feedback form is given in Figure 3.18. It

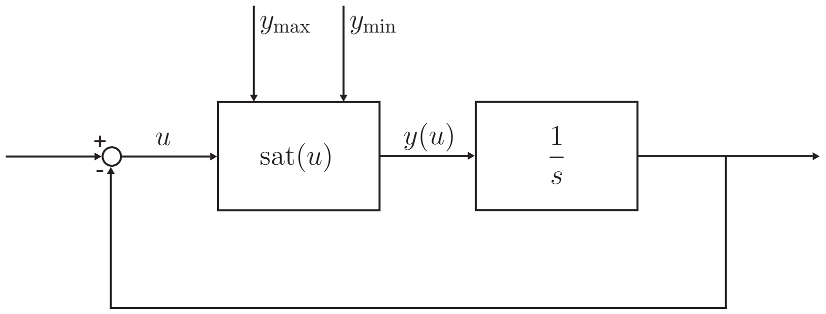


Figure 3.18: Rate limiter block diagram.

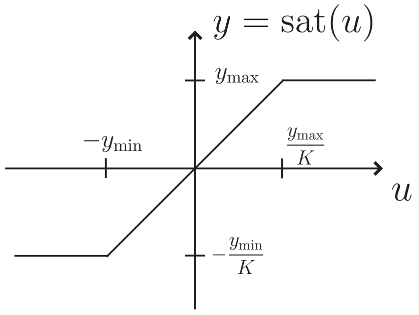


Figure 3.19: Graph of $y = \text{sat}(u)$.

is constructed by closing the loop around a saturation with gain K, $\text{sat}(u)$, and an integrator, $1/s$, in cascade. The inputs $y_{\max}$ and $y_{\min}$ represents the upper and lower bounds on the rate of change. The saturation is described mathematically by

$$\text{sat}(u) = \begin{cases} y_{\max} & Ku > y_{\max}, \\ Ku & -y_{\min} \leq Ku \leq y_{\max}, \\ -y_{\min} & Ku < -y_{\min}. \end{cases}$$

Its graph is shown in Figure 3.19. By choosing $y_{\max} = \infty$ and updating $y_{\min}(t) = 1/h'_{k'}$, for $t_{k'} \leq t < t_{k'+1}$, the rate of change constraint on $f(t)$ is taken into account in the model. In theory the gain K should be chosen as large as possible, ideally $K \to \infty$. In practice, however, a large K might introduce numerical problems when simulating the model.

By modifying $y_{\max}$, possible non-neglectable constraints on flight size increments can be modeled. Traffic shaping in terms of smoothing out abrupt window changes can be incorporated in the model as well, e.g., by filtering $w_f(t)$ before feeding it into the rate limiter.

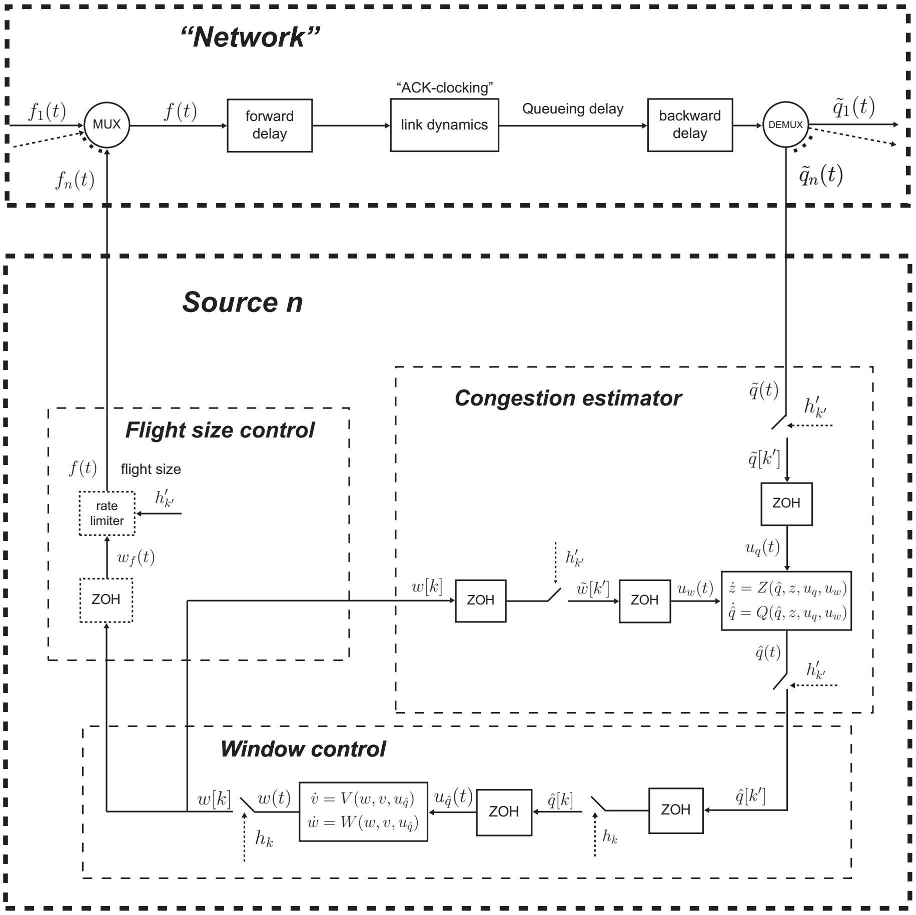


Figure 3.20: Network model.

3.5 Model summary

The model derived in the previous sections of this chapter is summarized in Figure 3.20. Sources' flight sizes are the input to the "network" which consist of the ACK-clocking model derived in Section 3.3 and propagation delay. The output from the "network" is the disturbed queuing delays which sources observe. A source n samples and creates an estimate of the queuing delay, this procedure is represented by the Congestion estimator. This was discussed in Section 3.4.1. The window size is updated in the Window control which uses the queuing delay estimate as input, see Section 3.4.2. The flight size control tries to equal the flight size, the actual packets in flight, and the window size variable whenever possible, see Section 3.4.3

for details. We remark that if the the window control sampling rate h_k is a multiple of the estimator sampling rate $h'_{k'}$, the right ZOH block in the window control in the figure is redundant and should be neglected.

3.6 Modeling FAST TCP

In order to give a concrete illustration of the modeling framework previously developed in this chapter we will now model the recently proposed time based TCP sibling FAST TCP (Jin *et al.*, 2004). The obtained model will serve as an illustration for the model simplification and the analysis techniques developed in Chapter 4 (see Section 4.4).

3.6.1 Introduction to FAST TCP

We will start with describing the algorithm.

Protocol rationale

Designed for large distance high speed data transfers FAST TCP adopts the delay-based approach of TCP Vegas and uses end-to-end queuing delay as aggregated price. The algorithm estimates this quantity and tries to stabilize the window size such that a targeted number of packets are buffered in the network. This strategy implies that the equilibrium that FAST is able to attain is efficient in the sense that full utilization of network resources is guaranteed. Furthermore, the design of FAST TCP is such that the equilibrium rate of a source is *independent* of the network propagation delay. This is expressed by the equilibrium constraint

$$x_n\, q_n = \alpha_n, \quad n = 1, \dots, N, \tag{3.18}$$

where $\alpha_n \in \mathbb{Z}^+$ is a protocol design parameter that corresponds to the number of packets the source tries to buffer in the network.

Window and estimator update in FAST TCP

Below, the basic features of the FAST TCP algorithm are briefly described. We will drop the subscript n for ease of notation.

FAST TCP is window based and applies standard ACK-clocking to adjust data packet transmission. The sending rate of FAST TCP is thus implicitly adjusted via the congestion window mechanism. Each sender updates its window size in discrete time according to

$$w[k+1] = (1-\gamma)w[k] + \gamma \frac{d}{d + \hat{q}[k]} w[k] + \gamma\alpha. \tag{3.19}$$

This update is performed *once* every RTT. The protocol parameter γ adjusts the gain of the algorithm, α was introduced in (3.18) and the input signal $\hat{q}$ is an estimate of the queuing delay along the path.

The observed aggregate queuing delay $\tilde{q}[k]$ can be approximated by subtracting the latency d from the measured RTT. However, this gives a noisy measurement of the "true" queuing delay that reflects the level of congestion in the network, it is therefore *estimated* by the source.

The estimate $\hat{q}[k]$ is formed from queuing delay samples measured at *each* acknowledgment arrival, $w[k]$ times for the kth update of (3.19). Denote discrete time at this time scale by k'. With the obvious abuse of notation, we refer to values of the queuing delay samples at this sampling time as $\tilde{q}[k']$ etc.. The estimator is

$$\hat{q}[k'+1] = (1-\sigma[k'])\,\hat{q}[k'] + \sigma[k']\tilde{q}[k'], \tag{3.20a}$$
$$\sigma[k'] = \min\{\kappa/w[k'], \nu\}\,. \tag{3.20b}$$

This non-linear filter has the characteristic of a low-pass filter, with a dynamic time constant of τ/κ. In the current implementation the filter parameters are $\kappa = 3$ and $\nu = 1/4$.

3.6.2 Model

The congestion control mechanism of FAST TCP fits into the algorithm structure defined by Figure 3.12, Figure 3.13, Figure 3.15 and Figure 3.17. In fact, since there are no internal states in any of the estimator and window update laws, we can identify $\mathcal{W}(w[k], \hat{q}[k])$ and $\mathcal{Q}(\hat{q}[k'], w[k'])$ as the right hand sides of (3.19) and (3.20) respectively. Furthermore, since the window is updated per round trip time and the estimator at every ACK arrival, the window update sampling time h_k corresponds to round trip times, and the estimator sampling time $h'_{k'}$ represents ACK inter-arrival times. The two sampling rates are related by

$$h_k = \sum_{k'=1}^{w_n[k]} h'_{k'+k'_k} \tag{3.21}$$

where $k'_k = \sum_{i=1}^{k-1} w_n[k]$ is the value of k' just before the start of the kth RTT. Note that due to $w_n[k]$ in (3.21), the relation between the two sampling times is time varying.

We observe that both (3.19) and (3.20) are special cases of (3.13). It is therefore straightforward to derive continuous time equivalent controls $W(w, u_{\hat{q}})$ and $Q(\hat{q}, u_q, u_w)$ (cf., Figure 3.16 and Figure 3.14) by making use of (3.16) and (3.17).

Start with the window control: we have from (3.19) that

$$w[k+1] = \mathcal{W}_1(\hat{q}[k])w[k] + \mathcal{W}_0$$

with

$$\begin{aligned}\mathcal{W}_1(\hat{q}[k]) &= 1-\gamma\frac{\hat{q}[k]}{d+\hat{q}[k]},\\ \mathcal{W}_0 &= \gamma\alpha;\end{aligned}$$

now (3.16) and (3.17) gives

$$\begin{aligned}W_1(u_{\hat{q}}) &= \frac{\log\left(\mathcal{W}_1(u_{\hat{q}})\right)}{h_k} = \frac{\log\left(1-\gamma\xi\right)}{h_k},\\ W_0(u_{\hat{q}}) &= \frac{\mathcal{W}_0\log(\mathcal{W}_1(u_{\hat{q}}))}{h_k(\mathcal{W}_1(u_{\hat{q}})-1)} = -\frac{\alpha\log\left(1-\gamma\xi\right)}{h_k\xi},\end{aligned}$$

where $\xi = u_{\hat{q}}/(d+u_{\hat{q}})$, which gives the continuous control law

$$\dot{w}(t) = \frac{\log\left(1-\gamma\xi(t)\right)}{h_k(t)}\left(w(t)-\frac{\alpha}{\xi(t)}\right). \tag{3.22}$$

Similarly for the estimation algorithm,

$$\hat{q}[k'+1] = \mathcal{Q}_1(w[k'])\hat{q}[k'] + \mathcal{Q}_0(\tilde{q}[k'], w[k'])$$

where

$$\begin{aligned}\mathcal{Q}_1(w[k']) &= 1-\sigma(w[k']),\\ \mathcal{Q}_0(\tilde{q}[k'], w[k']) &= \sigma(w[k'])\tilde{q}[k'];\end{aligned}$$

thus

$$\begin{aligned}Q_1(u_w) &= \frac{\log\left(\mathcal{Q}_1(u_w)\right)}{h'_{k'}} = \frac{\log\left(1-\sigma(u_w)\right)}{h'_{k'}},\\ Q_0(u_q, u_w) &= \frac{\mathcal{Q}_0\log(\mathcal{Q}_1(u_w))}{h'_{k'}(\mathcal{Q}_1(u_w)-1)} = -\frac{q[k']\log\left(1-\sigma(u_w)\right)}{h'_{k'}};\end{aligned}$$

and finally

$$\dot{\hat{q}}(t) = \frac{\log\left(1-\sigma(t)\right)}{h'_{k'}}\left(\hat{q}(t)-u_q(t)\right) \tag{3.23}$$

where $\sigma(t) = \min\{\kappa/u_w(t), \nu\}$.

We illustrate the equivalence of the continuous time model at sampling instants with an example.

Example 3.6.1. *Consider the FAST TCP window control update mechanism (3.19). Protocol parameters are set to $\alpha = 100$ and $\gamma = 0.5$. The algorithm inputs, the queuing delay estimates $\hat{q}[k]$ and the sample intervals h_k, are modeled as uniformly distributed random variables between zero and one, that is $\hat{q}[k], h_k \in U(0,1)$. The circles in Figure 3.21 shows the window size when the discrete window update law (3.19) is simulated. The solid line corresponds to the window size when the continuous time equivalent model of Figure 3.16 with (3.22) is simulated. We observe that the two simulations are more or less equivalent at sampling instants.* □

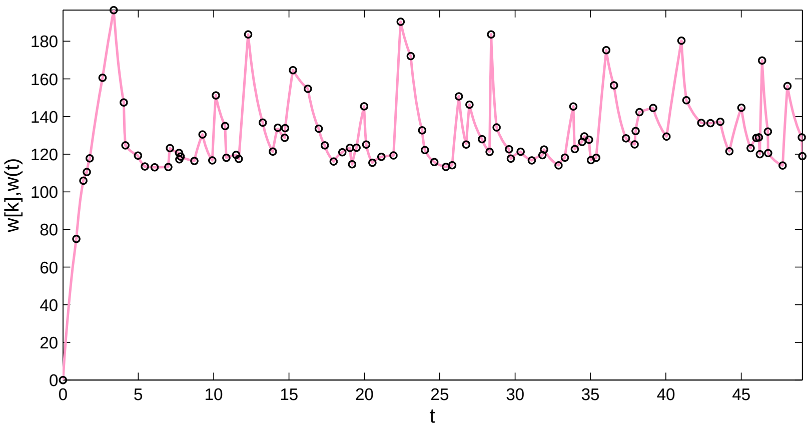


Figure 3.21: Window control simulation. Circles: $w[k]$, discrete window update (3.19). Solid line: $w(t)$, equivalent continuous time window control (3.22).

3.7 Summary

In this chapter we have shown how to derive deterministic fluid flow models for window based congestion control protocols using delay as network congestion notification.

We have seen that the system consists of an inner and an outer loop, where the inner loop is due to the so called ACK-clocking mechanism and the outer loop consists of the window control together with estimation procedures and traffic shaping components. A detailed model of the, to window based protocols, generic ACK-clocking mechanism was derived and guidelines for finding suitable continuous time models of the outer loop was provided. Finally we used these guidelines to model a specific protocol, namely FAST TCP.

3.8 Related work

Following the seminal work by Kelly *et al.* (1998) there are many studies on network dynamics. Network fluid flow models, where packet level information is discarded and traffic flows are assumed to be smooth in space and time appears in, e.g., (Hollot *et al.*, 2001a; Johari and Tan, 2001; Low *et al.*, 2002a; Baccelli and Hong, 2002; Altman *et al.*, 2004; Ying *et al.*, 2006), see also the book (Srikant, 2004).

The validity of results concerning dynamical properties rely heavily on the accuracy of the models. Considering window based congestion control, models with fundamentally different dynamical properties have been used to model the ACK-clocking dynamics (often referred to as "the link" in the literature). The most commonly appearing model can be found in, e.g., (Hollot *et al.*, 2001a; Low *et al.*,

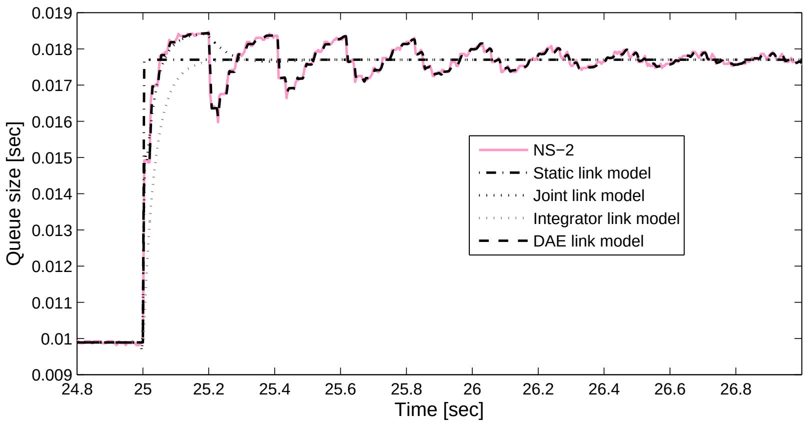


Figure 3.22: ACK-clocking dynamics for two flows sending over a single bottleneck link. Step response simulation. Basic configuration: Link capacity $c = 150\,\mathrm{Mbit/s}$. Packet size $\rho = 1040\,\mathrm{byte}$. Flow 1 propagation delay $d_1 = 10\,\mathrm{ms}$. Flow 2 propagation delay $d_2 = 190\,\mathrm{ms}$. Flow 1 window $w_1 = 210\,\mathrm{packets}$. Flow 2 window $w_2 = 1500\,\mathrm{packets}$. After convergence, at 25 seconds the first flows window is increased step-wise from 210 to 300 packets.

2002a; Baccelli and Hong, 2002; Altman *et al.*, 2004). Here an "integrator" link model is used, integrating the approximate excess rate on the link, and sources' rates are modeled as being equal to the window size divided by the RTT. On the other hand, in (Wang *et al.*, 2005; Wei *et al.*, 2006) transients are ignored due to the "self-clocking" effect and a "static" link model is proposed. In (Jacobsson *et al.*, 2006) a "joint" link model combining the immediate and long term integrating effect is proposed. This model is slightly refined in (Möller, 2008). The plot in Figure 3.22 demonstrates the limited accuracy of these models compared to the ACK-clocking model (3.10) derived in this thesis. We consider the data from the packet level NS-2 simulation as "true". We observe a heavy oscillation in the queue size that all previous model fails to capture. (Note that the "Joint link model" refers to the model used by Möller (2008) in this case.) For details about the simulation see Example 5.2.1 in Chapter 5.

The key in the ACK-clock modeling is to express the instantaneous rates flowing in to the queue in terms of the flight sizes which is done in the integral equation (3.10c). Variants of this equation appears in the passing in, e.g., (Mo *et al.*, 1999) and (Eng, 2007) but is not pursued.

The focus of window based congestion control analysis and design is, naturally, the window control. An early fluid flow model of the window control for the standard AIMD algorithm used by TCP can be found in (Misra *et al.*, 2000). It is very similar to the model that was derived in Example 2.5.2. Considering window based congestion control protocols using queuing delay as congestion notification, models for, e.g., TCP Vegas can be found in (Mo *et al.*, 1999; Boutremans and Le Boudec, 2000; Low *et al.*, 2002c). The window update model that the design of FAST TCP is based on appears in (Wang *et al.*, 2005) and (Wei *et al.*, 2006). A model of the actual implementation taking sampling effects and estimator dynamics into account was introduced in (Jacobsson *et al.*, 2008).

Over the last fifteen years or so there has been significant progress in modeling and identification when the model is to be used for control design. The present state-of-the art is summarized in (Hjalmarsson, 2005, 2003), see also (Albertos and Sala Piqueras, 2002). For a comprehensive treatment on system identification in general, see the book (Ljung, 1999).

For an introduction to sampling and discrete time systems we refer to (Åström and Wittenmark, 1997).

Chapter 4

Congestion control analysis

TRACTABLE models allow for mathematical analysis as an efficient route to reveal properties about the system of interest. In this chapter we will use the models derived in the previous chapter to study window based congestion control. We have learned that the window based system is composed of two loops, recall Figure 3.4. To characterize the properties of the outer loop, the window control and estimator dynamics, a comprehensive understanding of the properties of the inner loop, the ACK-clocking mechanism, is essential. In line with this, the first contribution of this chapter is an equilibrium analysis of the ACK-clocking mechanism. Based on this we move on to consider the entire system. In order to be able to guarantee an efficient resource allocation policy of a protocol and that the network operates in a favorable region of the state space, it is important that the system equilibrium is unique and, furthermore, that the window control is closed loop stable. This will be investigated for the recently proposed congestion control protocol FAST TCP.

4.1 Introduction

Before continuing with the specific analysis let us introduce some tools that will be useful.

4.1.1 Equilibrium characterization

It was shown in Section 2.5.3 that a quite large class of congestion control algorithms can be interpreted as a distributed solver to a convex optimization problem. We saw that this optimization perspective on network congestion control can be used as a means to establish the uniqueness of the equilibrium of a networked dynamical system, and, moreover, relate the equilibrium to specific flow level properties of interest such as fairness.

In this part we will specify in more detail how to link the equilibrium of a network with window based congestion control protocols using queuing delay as congestion signal to the solution of a utility maximization problem.

The network utility maximization problem

We will start with a brief review of the network utility maximization problem introduced in Section 2.5.3.

Let $\bar{x}_i$ be the sending rate of the ith source, $i = 1, \ldots, \bar{N}$. Furthermore, assume that sender i associates a utility $\bar{U}_i(\bar{x}_i)$ with a particular sending rate $\bar{x}_i$, and that $\bar{U}_i(\bar{x}_i)$ is a concave function. Let $\bar{y}_\ell$ denote the aggregated rate, i.e., the sum of individual rates, at link ℓ, and $\bar{c}_\ell$ the capacity of that link, $\ell = 1, \ldots, \bar{L}$. Let $\bar{R}$ be the routing matrix and recalling its definition we have $\bar{y} = \bar{R}\bar{x}$. The network utility maximization problem, that maximizes the total utility without violating network capacity constraints, can in this context be formulated as

$$\begin{aligned} \max_{\bar{x} \geq 0} \quad & \sum_{i=1}^{\bar{N}} \bar{U}_i(\bar{x}_i), \\ \text{s.t.} \quad & \bar{R}\bar{x} \leq \bar{c}, \end{aligned}$$

where inequalities are element-wise. This is a convex optimization problem for which strong duality holds since the convex constraint set has a non-empty interior. Thus, we know from convex optimization (Boyd and Vandenberghe, 2004) that it exists a global optimal solution to this problem which, furthermore, is characterized by the Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} &\bar{x} \geq 0, && \bar{p} \geq 0, && \text{non-negativity constraint;} \\ &\bar{y} \leq \bar{c}, && && \text{capacity constraint;} \\ &\bar{y} = \bar{R}\bar{x}, && \bar{q} = \bar{R}^{\mathrm{T}}\bar{p}, && \text{aggregation;} \\ &\bar{q}_i = \bar{U}_i'(\bar{x}_i), && && \text{gradient condition;} \\ &0 = \bar{p}^{\mathrm{T}}(\bar{c} - \bar{y}), && && \text{complementary slackness;} \end{aligned}$$

for $i = 1, \ldots, \bar{N}$. If $\{\bar{U}_i\}$ are strictly concave the optimal rates $\bar{x}_i$ are unique. In addition, assume there exist two different optimal vectors of dual variables $\bar{p}$ and $\tilde{p}$, then

$$\bar{R}^T(\bar{p} - \tilde{p}) = \bar{q} - \bar{q} = 0.$$

If $\bar{R}$ has full row rank, then the columns of $\bar{R}^T$ are linearly independent and thus $\bar{p} = \tilde{p}$. Therefore, if $\bar{R}$ has full row rank, the dual variables $\bar{p}_\ell$ are unique as well.

"Reverse-engineering"

Consider a network that consists of links that have infinitely large buffers and which applies FIFO scheduling. The network is utilized by window based sources that uses the observed total queuing delay as a measure of the amount of congestion in the network. Next we will associate the equilibrium of such a system with the solution to the utility maximization problem previously discussed.

In a FIFO buffering network with infinite queues we know from Section 3.3 that the dynamics of each buffer is given by

$$\dot{b}_l(t) = \begin{cases} \frac{1}{c_l}\left(\sum_{n=1}^{N} r_{l,n} x_{l,n}(t) - c_l\right), & \text{if } b_l(t) > 0 \text{ or } \sum_{n=1}^{N} r_{l,n} x_{l,n}(t) - c_l > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where the instantaneous queue input rates $x_{l,n}(t)$ are defined in the ACK-clocking model (3.10). It is assumed that a source n adjust its window size according to the dynamical law

$$\dot{w}_n(t) = W_n(w_n(t), \hat{q}_n(t)). \tag{4.2}$$

Let us assume the existence of an equilibrium point (b, w) characterized by letting $\dot{b}_l = 0$ and $\dot{w}_n = 0$ in (4.1) and (4.2), and by $b_l \geq 0, w_n \geq 0$. The non-negativity constraints are required since buffer sizes and windows are non-negative by definition. Note that in equilibrium there is no difference between windows and flight sizes, so $w_n = f_n$, and the queue input rates are equal along path n, i.e., $x_{l,n} = x_{k,n} =: x_n$ for all feasible k, l. Furthermore, signaling delays can be ignored implying that the aggregate equilibrium rate on each link can be expressed as $y = Rx$. The aggregated queuing delay along a path is the sum of the equilibrium queues on the path, therefore $q = R^{\mathrm{T}} b$. Subsequently the equilibrium round trip times can be written $\tau_n = d_n + q_n$, and the rates $x_n = w_n/\tau_n$, for all $n = 1, \ldots, N$. Furthermore assume that sources' queuing delay estimates are unbiased such that $\hat{q}_n = q_n$ holds for all $n = 1, \ldots, N$.

Let us consider the implication of the buffer equilibrium condition $\dot{b}_l = 0$. It is realized from (4.1) that in equilibrium either

$$b_l = 0 \quad \text{and} \quad \sum_{n=1}^{N} r_{l,n} x_{l,n} = \sum_{n=1}^{N} r_{l,n} x_n = y_l \leq c_l$$

or

$$\sum_{n=1}^{N} r_{l,n} x_{l,n} = \sum_{n=1}^{N} r_{l,n} x_n = y_l = c_l$$

holds. The equivalent condition in vector notation is: $y \leq c$ and $b^{\mathrm{T}}(c - y) = 0$.

Similarly, for the window control, $\dot{w}_n = 0$ and (4.2) gives $W_n(w_n, q_n) = 0$.

In summary, the equilibrium conditions can be written as

$$\begin{aligned} x &\geq 0, & b &\geq 0, \\ y &\leq c, & & \\ y &= Rx, & q &= R^{\mathrm{T}} b, \\ x_n &= \frac{w_n}{d_n + q_n}, & 0 &= W_n(w_n, q_n), \\ 0 &= b^{\mathrm{T}}(c - y), & & \end{aligned}$$

Note the similarity between these conditions and the KKT conditions for the network utility maximization problem. We would now like to pose some assumptions on the window control function $W_n(w_n, q_n)$, that determines the equilibrium relation between the window size w_n and the queuing delay q_n, such that the equilibrium conditions summarized above coincides with the solution of a utility maximization problem with routing R, capacities c and strictly concave source utility functions $U_n(x_n)$. This would imply the uniqueness of the equilibrium rates x, and if R has full rank that queuing delays b are unique as well.

First, assume that window control is such that we can solve $W_n(w_n, q_n) = 0$ with respect to w_n, then we can write

$$w_n = Q_n(q_n).$$

This implies that the equilibrium rate is

$$x_n = \frac{w_n}{d_n + q_n} = \frac{Q_n(q_n)}{d_n + q_n} =: F_n(q_n).$$

Under the assumption that $F_n(q_n)$ is strictly decreasing its inverse function exists,

$$q_n = f_n(x_n),$$

which is also a strictly decreasing function. Let us define the source utility function as

$$U_n(x_n) = \int f_n(x_n) dx_n, \tag{4.3}$$

and note that since $f_n(x_n)$ is strictly decreasing by assumption $U_n(x_n)$ is strictly concave. Now we have by definition and under the given assumptions that the equilibrium conditions

$$x_n = \frac{w_n}{d_n + q_n}, \qquad 0 = W_n(w_n, q_n),$$

are equivalent to

$$q_n = f_n(x_n) = U_n'(x_n).$$

This observation leads to the conclusion that the equilibrium rates x and queuing delays b are unique and completely characterized of the KKT conditions of a utility maximization problem defined by the R, c and U_n. To guarantee the uniqueness of the equilibrium windows w we furthermore need to impose that the functions $Q_n(q_n)$ are strictly monotone.

Note that we until now for simplicity have assumed that no non-window based cross traffic is present. This is, however, easily adjusted for by replacing the capacity c with the available bandwidth $c - x_c$ in the equilibrium conditions.

Studying the utility function of a protocol is many times a convenient way to deduce the uniqueness of the equilibrium. This strategy will be used in Section 4.2.1 in the study of the ACK-clocking mechanism and in Section 4.4.1 in the analysis of FAST TCP.

4.1.2 Stability

In dealing with the stability of feedback systems, the machinery of control theory is suitable. Below we will present some standard results that we will use as well as a lemma that is tailored for our needs.

The $\mathcal{H}_\infty$ space

Consider the stability of a linear system represented by the rational open loop transfer function $L(s) = P(s)/Q(s)$. Assuming there are no common roots to the polynomials P and Q this system is stable if and only if there exist no complex right half plane poles p of $L(s)$, i.e., all solutions to the characteristic equation $Q(p) = 0$ satisfies $\operatorname{Re} p < 0$. This condition is trivial to check when $Q(s)$ is of first and second order, but, naturally, gets more involved as the polynomial order grows.

For the more general case when $P(s)$ and $Q(s)$ are analytic functions (i.e., $L(s)$ is meromorphic) the issue of stability gets more intricate. The reason is that it for this case may exist infinitely many zeros and/or poles and possibly accumulation points. For such systems we will instead of studying the poles explicitly establish stability by showing that the transfer function is in $\mathcal{H}_\infty$, the space of functions that are analytic and bounded in the open right-half complex plane. Transfer functions in $\mathcal{H}_\infty$ are input output stable which also implies asymptotic stability. The definition of $\mathcal{H}_\infty$ is as follows. Let $\mathbb{C}^+$ be the open right half plane, $\{z : \operatorname{Re}(z) > 0\}$, and $\bar{\mathbb{C}}^+$ be its closure, $\{z : \operatorname{Re}(z) \geq 0\}$.

Definition 4.1.1. *A function $G : \bar{\mathbb{C}}^+ \to \mathbb{C}^{n\times m}$ is in $\mathcal{H}_\infty$ if*

(a) *$G(s)$ is analytic in $\mathbb{C}^+$;*

(b) *for almost every real number ω,*

$$\lim_{\sigma\to 0^+} G(\sigma + j\omega) = G(j\omega);$$

(c) $\sup_{s\in\bar{\mathbb{C}}^+} \bar{\sigma}(G(s)) < \infty$,

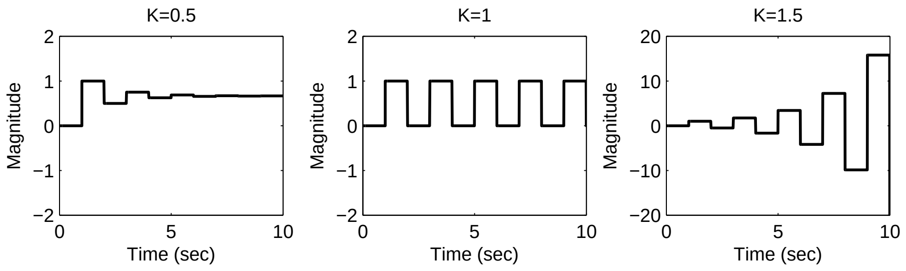


Figure 4.1: Simulation of (4.5) for different values of K.

where $\bar{\sigma}$ denotes the largest singular value.

As an illustration of the use of the theory, let us study a concrete example.

Example 4.1.1. *Consider the non-rational but analytic transfer function*

$$G(s) = \frac{1}{1 + Ke^{-s}}, \tag{4.4}$$

where K is a strictly positive gain parameter. This correspond to the time domain system

$$y(t) + Ky(t-1) = u(t). \tag{4.5}$$

At sampling instants it equals the time discrete system

$$y_k + Ky_{k-1} = u_k, \tag{4.6}$$

known to be asymptotically stable when its poles are located inside the unit circle. This is the case when $|K| < 1$. The plots in Figure 4.1 shows simulations of (4.5) for $K = 0.5$ (stable), $K = 1$ (marginally stable), and $K = 1.5$ (unstable). They confirm our prediction of when the system becomes unstable. We will now exam the stability of (4.5) analytically.

Let $s = \sigma + j\omega$ when $\sigma \in \mathbb{R}, \omega \in \mathbb{R}$, so that

$$G(\sigma + j\omega) = \frac{1}{1 + Ke^{-\sigma}(\cos(\omega) + j\sin(\omega))}. \tag{4.7}$$

Obviously the imaginary part is zero for $\omega = \bar{\omega}_n = n\pi, n = \pm 0, 1, 2, \dots,$ only, and it follows that

$$|G(\sigma + j\omega_n)| < \infty, \quad \textit{for all} \quad \omega \neq \bar{\omega}.$$

Furthermore we have

$$G(\sigma + j\bar{\omega}_n) = \frac{1}{1 + (-1)^n Ke^{-\sigma}}.$$

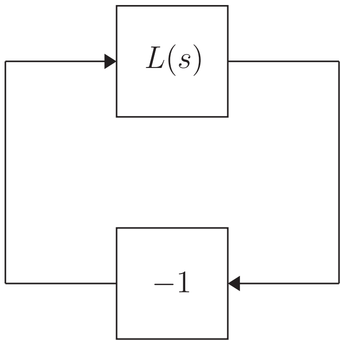


Figure 4.2: Closed loop system.

Clearly for even n this expression is bounded since the denominator is bounded below by 1 for all $K > 0$. For odd n, however,

$$\lim_{\omega \to \bar{\omega}_n} |G(\sigma + j\omega)| = \infty$$

if

$$Ke^{-\sigma} = 1 \quad \Leftrightarrow \quad \sigma = \log(K).$$

We observe that for $K \geq 1$ only, we have $\sigma \geq 0$. Thus, for that case, $G(s)$ is not in $\mathcal{H}_\infty$ since it is unbounded in the right-half plane which violates condition (c) in Definition 4.1.1. In summary, we have seen that the system is stable for all positive $K < 1$. This is consistent with our preliminary analysis based on linear systems theory of discrete systems and simulations.

Closed loop stability using the Nyquist criterion

Considering closed loop stability of a system with open loop gain $L(s)$, see Figure 4.2, an often convenient alternative to explicitly study the location of the poles of the closed loop transfer function is to apply the Nyquist criterion. Under the assumption that $L(s)$ is open loop stable the simplified Nyquist criterion presented below provides necessary and sufficient conditions for closed loop stability.

Theorem 4.1.1 (The simplified Nyquist criterion)**.** *Assume that $L(s)$ does not have any poles in the closed right complex half plane. Then the closed loop system is asymptotically stable if and only if the contour $L(j\omega)$, $0 \leq \omega < \infty$ does not cross or encircle -1.*

Remark that the Nyquist criterion also allows for non-rational transfer functions. Let us continue the previous example.

Example 4.1.2 (Example 4.1.1 continued)**.** *The poles of $G(s)$, defined by (4.4), is given by the characteristic equation*

$$1 + Ke^{-s} = 0.$$

This is also the characteristic equation for a closed loop system with open loop gain $L(s) = Ke^{-s}$. The contour of $L(j\omega) = Ke^{-j\omega}, 0 \leq \omega < \infty$, i.e., the Nyquist curve, is a circle centered at the origin with radius K. Obviously it will encircle -1 for $K > 1$, touch -1 for $K = 1$, and not encircle for $0 < K < 1$. From the simplified Nyquist criterion we thus can conclude that the system is stable for $0 < K < 1$, perfectly in line with our previous findings.

Exploring model structure

It turns out that when analyzing the stability of some congestion control algorithms, such as FAST TCP, the loop gain has the structure of a weighted sum of individually stable transfer functions which shares a common pole located at the point equal to the sum of some non-negative weights l_i,

$$L(s) = \sum_{n=1}^{N} l_n \frac{L_n(s)}{s + \sum_{i=1}^{N} l_i}. \tag{4.8}$$

Depending on the complexity of $\{L_n(s)\}$ and the number of elements in the sum it may be a formidable task to analyze the full loop gain $L(s)$ explicitly. However, the following lemma, which utilizes the Nyquist criterion, gives conditions on each $L_n(s)$ that is sufficient to guarantee closed loop stability.

Lemma 4.1.2. *Denote the half plane under the line that passes $-1 + j0$ with slope $\beta/(\omega\hat{\tau})$, where $\beta > 0$, by*

$$H(\omega) = \left\{ z \mid \arg(z+1) - \arctan\left(\frac{\beta}{\omega\hat{\tau}}\right) \in (-\pi, 0) \right\}. \tag{4.9}$$

Let $F_n(s)$, $n = 1, \ldots, N$, be stable transfer functions, and let

$$0 < -\min_{\theta > 0} \mathrm{Re}(F_n(j\theta)) < 1/\gamma_n.$$

Then a system with open loop gain

$$L(j\omega) = \beta \sum_{n=1}^{N} \frac{\mu_n}{\tau_n} \frac{\gamma_n F_n(j\omega)}{j\omega + \beta/\hat{\tau}} \tag{4.10}$$

where $1/\hat{\tau} \equiv \sum_{n=1}^{N} \mu_n/\tau_n$, satisfies $L(j\omega) \in H(\omega)$ for all ω, and is closed loop stable for any $\tau_n > 0, \mu_n > 0$, $n = 1, \ldots, N$.

Proof. By definition, $L(j\omega) \in H(\omega)$ is equivalent to

$$\arg(L(j\omega)+1) - \arctan\left(\frac{\beta}{\omega\hat{\tau}}\right) \in (-\pi, 0). \tag{4.11}$$

Substituting (4.10) and noting that

$$\arg\left(j\omega + \frac{\beta}{\hat{\tau}}\right) + \arctan\left(\frac{\beta}{\omega\hat{\tau}}\right) = \arctan\left(\frac{\omega\hat{\tau}}{\beta}\right) + \arctan\left(\frac{1}{\omega\hat{\tau}/\beta}\right) = \frac{\pi}{2},$$

condition (4.11) can be further rewritten as

$$\arg\left(\beta\sum_{n=1}^{N}\frac{\mu_n}{\tau_n}\gamma_n F_n(j\omega) + j\omega + \frac{\beta}{\hat{\tau}}\right) \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

which is equivalent to

$$\beta \mathrm{Re}\left(\sum_{n=1}^{N}\frac{\mu_n}{\tau_n}\gamma_n F_n(j\omega) + j\frac{\omega}{\beta} + \frac{1}{\hat{\tau}}\right) > 0.$$

Since $\mathrm{Re}\left(\gamma_n F_n(j\omega)\right) > -1$ by hypothesis, and furthermore $1/\hat{\tau} = \sum_{n=1}^{N}\mu_n/\tau_n$ by definition, it is established that $L(j\omega) \in H(\omega)$ for all ω.

Thus, since

$$(-\infty, -1] \cap \bigcup_{\omega \geq 0} H(\omega) = \emptyset,$$

the Nyquist curve for L cannot encircle -1. The stability of $F_n(s)$ implies that $L(j\omega)$ is open loop stable, and hence the system is closed loop stable by the Nyquist criterion. □

The construction used for Lemma 4.1.2 is depicted in Figure 4.3 for $\beta = 1$, $\tau_1 = 1$, $\tau_2 = 5$, $\mu_1 = \mu_2 = 1/2$, at $\tilde{\omega}\hat{\tau} = 3$, and where $F_n(s)$ is such that $L(j\omega) \in H(\omega)$. It illustrates how we need to consider the full open loop transfer function. The set $H(\tilde{\omega})$ is the set below the line with slope 1/3. While the individual parts $\tilde{F}_n(j\tilde{\omega})$ of the weighted sum in the open loop transfer function $L(j\tilde{\omega}) = \sum_n \mu_n \tilde{F}_n(j\tilde{\omega})$, cf., (4.10), may not be in this set, the full loop gain $L(j\tilde{\omega})$ is.

4.2 ACK-clocking

This section analyze the properties of the ACK-clocking mechanism. In particular it shows the uniqueness of the equilibrium of the general ACK-clocking model (3.10):

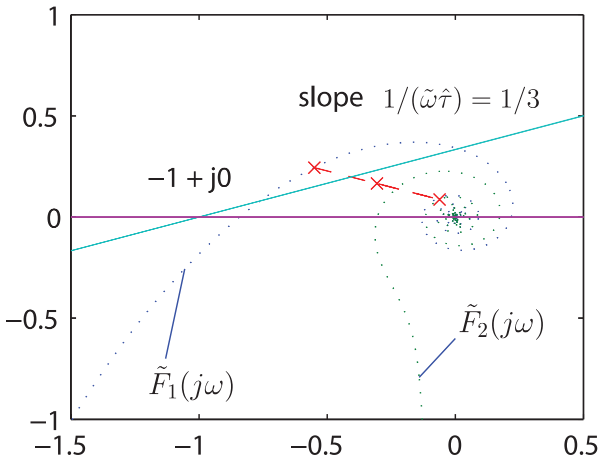


Figure 4.3: An example of a line of slope $1/(\tilde{\omega}\hat{\tau})$ which bounds $L(j\tilde{\omega})$ at a particular frequency $\tilde{w}$, denoted by the center cross. Note that the individual terms $\tilde{F}_n(j\tilde{\omega})$, denoted by the individual crosses, are not all below this line, however the weighted sum is.

for all $l = 1, \dots, L$,

$$\begin{aligned}
c_l \dot{b}_l(t) &= \sum_{n=1}^{N} r_{l,n} x_{l,n}(t) + x_{c;l}(t) - c_l, \\
f_{l,n}(t + \tau_{l,n}^f(t)) &= f_n(t), \qquad n = 1, \dots, N, \\
\int_t^{t+\tau_{l,n}(t)} x_{l,n}(s) ds &= f_{l,n}(t + \tau_{l,n}(t)), \qquad n = 1, \dots, N, \\
\hat{\tau}_{l,n,i}(t) &= \vec{d}_{l,n,i} + \sum_{k=1}^{i-1} \vec{b}_{l,n,k}(t + \hat{\tau}_{l,n,k}(t)), \qquad n = 1, \dots, N, \\
\tau_{l,n}(t) &= d_n + \sum_{i=1}^{L_n} \vec{b}_{l,n,i}(t + \hat{\tau}_{l,n,i}(t)), \qquad n = 1, \dots, N, \\
\tau_{l,n}^f(t) &= \hat{\tau}_{\ell(n),n,m(l,n)+1}(t), \qquad n = 1, \dots, N.
\end{aligned}$$

To recall details about the model see the derivation of (3.10) given in Section 3.3.5. It is also, for the single link case, i.e., $l = 1$ in the model above, proved that the system is asymptotically stable from flight sizes to queuing delays but that at the same time, the rates may have sustained oscillations.

4.2.1 Equilibrium

ACK-clocking can be interpreted as a congestion control algorithm applied at the source, with queuing delay b as price signal p fed back from the network (i.e., $p = b$), and with tuning parameter f (flight size). In this context, we are able to apply the utility optimization framework to characterize the equilibrium in the following theorem. Note that the flight size f equals the window size w in equilibrium.

Theorem 4.2.1. *For given positive vectors w, d and c, the equilibrium rates x^* of the ACK-clocking model (3.10) are unique, and if R is full rank, then the queuing delays p^* are also unique.*

Proof. A feasible equilibrium point (x^*, p^*) satisfies $\sum_{n=1}^{N} R_{l,n} x_n^* \leq c_l$ for all l, n and

$$\begin{aligned} x_n^* &:= \frac{w_n}{d_n + q_n^*}, \\ q_n^* &= \sum_{l=1}^{L} R_{l,n} p_l^*, \\ p_l^* &\geq 0. \end{aligned}$$

The parameters w_n, d_n, c_l are fixed. The equilibrium point can be expressed as

$$\sum_{l=1}^{L} R_{l,n} p_l^* = q_n^* = \frac{w_n}{x_n^*} - d_n. \tag{4.12}$$

Let

$$U_n(x_n^*) = w_n \log(x_n^*) - d_n x_n^*$$

which is strictly concave. Note that (4.12) is the KKT condition to the convex program

$$\begin{aligned} \max_{x \geq 0} \quad & \sum_{n=1}^{N} U_n(x_n), \\ \text{s.t.} \quad & Rx \leq c, \end{aligned}$$

with convex feasible set with non-empty interior. Thus there exists a unique optimal solution x^*, see (Boyd and Vandenberghe, 2004), and by (4.12), a unique q^*. Assume there exist two optimal queuing delay vectors p^* and $\tilde{p}^*$, then

$$R^T(p^* - \tilde{p}^*) = q^* - q^* = 0.$$

If R has full row rank, then the columns of R^T are linearly independent and thus $p^* = \tilde{p}^*$. Therefore, if R has full row rank then the equilibrium (x^*, p^*) is unique. □

Linearization

In the next section we will analyze the local stability of the ACK-clocking model for the single bottleneck link case. As a preparation to this we will now linearize the single bottleneck ACK-clocking model (3.8), i.e.,

$$\begin{aligned} &\dot{b}(t) - \frac{1}{c}\left(\sum_{n=1}^{N} x_n(t) + x_c(t) - c\right) = 0, \\ &\int_t^{t+\tau_n(t)} x_n(s)\,ds - f_n(t+\tau_n(t)) = 0, \qquad n = 1,\dots,N, \end{aligned}$$

around the unique equilibrium (b, f, x, x_c). Consider small perturbations in variables around this equilibrium. Recall that $\tau_n(t) = d_n + b(t)$ and thus $\tau_n = d_n + b$. Define

$$H(z,\dot{z}) = \begin{pmatrix} \dot{b}(t) - \frac{1}{c}\left(\sum_{n=1}^{N} x_n(t) + x_c(t) - c\right) \\ (1+\dot{b}(t))\left(x_1(t+\tau_1(t)) - \dot{f}_1(t+\tau_1(t))\right) - x_1(t) \\ (1+\dot{b}(t))\left(x_2(t+\tau_2(t)) - \dot{f}_2(t+\tau_2(t))\right) - x_2(t) \\ \vdots \\ (1+\dot{b}(t))\left(x_N(t+\tau_N(t)) - \dot{f}_N(t+\tau_N(t))\right) - x_N(t) \end{pmatrix},$$

where variables are collected according to

$$z = \begin{pmatrix} b(t) & f^{\mathrm{T}}(t+\tau(t)) & x^{\mathrm{T}}(t+\tau(t)) & x^{\mathrm{T}}(t) & x_c(t) \end{pmatrix}^{\mathrm{T}},$$

and where $f(t+\tau(t))$ is a vector with nth element $f_n(t+\tau_n(t))$ and similarly for $x(t+\tau(t))$. In this notation

$$H(z,\dot{z}) = 0$$

is equivalent to (3.8). A linear approximative model valid for small perturbations δz around an equilibrium point z^* is obtained by a first order Taylor expansion:

$$0 = H(z,\dot{z}) \approx \frac{\partial H(z^*,0)}{\partial z}\delta z + \frac{\partial H(z^*,0)}{\partial \dot{z}}\delta \dot{z}.$$

Following the convention that time delays in variables' arguments are modeled by their equilibrium values (i.e., $x_n(t+\tau_n(t)) \approx x_n(t+\tau_n)$), and hence not considered in the linearization, yields the linear dynamics

$$\delta\dot{b}(t) - \sum_{n=1}^{N} \frac{\delta x_n(t)}{c} - \frac{\delta x_c(t)}{c} = 0, \tag{4.13a}$$

$$x_n \delta\dot{b}(t) - \delta\dot{f}(t+\tau_n) + \delta x_n(t+\tau_n) - \delta x_n(t) = 0, \tag{4.13b}$$

for $n = 1, \ldots, N$. Here δ preceding a variable denote small perturbations. Let upper-case $X(s)$ denote the Laplace transform of $x(t)$. Applying the Laplace transform to (4.13b) we can solve for the sources' queue input rates explicitly

$$\Delta X_n(s) = \frac{s}{e^{-s\tau_n} - 1} \left(x_n e^{-s\tau_n} \Delta B(s) - \Delta F_n(s) \right).$$

This expression can be used to eliminate the ΔX_n dependence in the frequency domain version of (4.13a)

$$s\Delta B(s) - \frac{1}{c} \sum_{n=1}^{N} \Delta X_n(s) - \frac{1}{c} \Delta X_c(s) = 0.$$

This gives that the linear ACK-clocking dynamics are described by

$$\left(c + \sum_{n=1}^{N} x_n \frac{e^{-s\tau_n}}{1 - e^{-s\tau_n}} \right) \Delta B(s) = \sum_{n=1}^{N} \frac{\Delta F_n(s)}{1 - e^{-s\tau_n}} + \frac{1}{s} \Delta X_c(s). \tag{4.14}$$

In the derivation of the single bottleneck link ACK-clocking model (3.8) we assumed zero forward propagation delay for simplicity. (This was however accounted for in the general model (3.10).) To model a non-zero forward propagation delay τ_n^f, simply multiply $\Delta F_n(s)$ by $e^{-s\tau_n^f}$ in (4.14).

The simplifying assumption ignoring variability in delay in variable arguments when linearizing, is standard in this type of work but was unfortunately not further motivated. However, the linear model is validated in Section 5.2.1 and performs well in the simulated scenarios. Furthermore, predictions based on the linear model appears to be accurate, see the analysis section that follows.

4.2.2 Stability

We will now investigate the stability of the ACK-clocking mechanism using the linear model (4.13) just derived.

Due to the conservation of packet principle of the ACK-clocking mechanism, i.e., a new packet is not injected into the network until an old packet leaves, the ACK-clock system is stable in the sense that the queues stay bounded for bounded flight sizes. This is realized as follows.

If we for simplicity assume no cross traffic, then the total amount of packets in the network at an instant of time is the sum of the sources' flight sizes. The packets that are not in transit between links are located in the queues. Since the flight sizes are bounded above by assumption, the maximum number of packets in the network is bounded and thus the number of packets in the queues must be upper bounded as well.

Let us now investigate the issue of stability in more detail. The following theorem shows the stronger result that the linearized single bottleneck dynamics (4.14) relating the flight sizes δf to the queue δb are asymptotically stable, ruling out persistent oscillations in these quantities.

Theorem 4.2.2. *For all* $0 < x_n \leq c$, $\tau_n > 0$, $n = 1, \ldots, N$, *and*

$$\sum_{n=1}^{N} x_n \leq c,$$

the function $G_{bf} : \bar{\mathbb{C}}^+ \to \mathbb{C}^{1\times N}$ *whose ith element is given by*

$$G_{bf_i}(s) = \frac{1}{(1-e^{-s\tau_i})\left(c + \sum_{n=1}^{N} x_n \frac{e^{-s\tau_n}}{1-e^{-s\tau_n}}\right)}, \tag{4.15}$$

is stable.

Proof. We know from Definition 4.1.1 that is sufficient to confirm that:

(a) $G_{bf}(s)$ is analytic in $\mathbb{C}^+$;

(b) for almost every real number ω,

$$\lim_{\sigma\to 0^+} G_{bf}(\sigma + j\omega) = G_{bf}(j\omega);$$

(c) $\sup_{s\in\bar{\mathbb{C}}^+} \bar{\sigma}(G_{bf}(s)) < \infty$

where $\bar{\sigma}$ denotes the largest singular value.

Conditions (a) and (b) are satisfied if they hold element-wise. Furthermore

$$\sup_{s\in\bar{\mathbb{C}}^+} \bar{\sigma}(G_{bf}(s)) = \sup_{s\in\bar{\mathbb{C}}^+} \sqrt{\bar{\lambda}(G_{bf}(s)G_{bf}^*(s))} = \sup_{s\in\bar{\mathbb{C}}^+} \sqrt{\sum_{i=1}^{N} G_{bf_i}(s)\overline{G}_{bf_i}(s)}$$
$$\leq \sum_{i=1}^{N} \sup_{s\in\bar{\mathbb{C}}^+} |G_{bf_i}(s)|.$$

Thus, condition (c) holds if

$$\inf_{s\in\bar{\mathbb{C}}^+} \left|G_{bf_i}^{-1}(s)\right| > 0 \text{ for all } i = 1, \ldots, N. \tag{4.16}$$

It is therefore sufficient to establish (a), (b) and (c) for each transfer function element $G_{bf_i}(s)$.

Start with the boundedness condition (c). It is sufficient to show that there is no sequence $s_l = \sigma_l + j\omega_l \in \bar{\mathbb{C}}^+$ with $\lim_{l\to\infty} |1/G_{bf_i}(s_l)| = 0$. This will be established by showing that the limit evaluated on any convergent subsequence is greater than 0. Consider a subsequence with $\sigma_l \to \sigma$, $\omega_l \to \omega$.

Case 1, $\sigma = \infty$:

$$1/G_{bf_i}(s_l) \to c > 0.$$

Case 2, $\sigma \in (0, \infty)$: By the triangle inequality,

$$|1 - e^{-s_l\tau_i}| \geq \left|1 - |e^{-s_l\tau_i}|\right| \to 1 - e^{-\sigma\tau_i} > 0. \tag{4.17}$$

Furthermore, for $\sigma > 0$, we have for an $A = e^{s\tau_n}, |A| > 1$,

$$\left(\mathrm{Re}\left(\frac{1}{A-1}\right) - \frac{1}{|A|^2-1}\right)^2 + \left(\mathrm{Im}\left(\frac{1}{A-1}\right)\right)^2 - \left(\frac{|A|}{|A|^2-1}\right)^2$$
$$= \left|\frac{1}{A-1}\right|^2 - \frac{2\mathrm{Re}(1/(A-1))}{|A|^2-1} - \frac{1}{|A|^2-1} = \frac{|A|^2 - 1 - (A + \bar{A} - 2) - |A-1|^2}{|A-1|^2\left(|A|^2-1\right)}$$
$$= \frac{A\bar{A} - 1 - A - \bar{A} + 2 - A\bar{A} + A + \bar{A} - 1}{|A-1|^2\left(|A|^2-1\right)} = 0.$$

Thus we can conclude that $1/(e^{s_l\tau_n} - 1)$ lies on the circle with center $1/(A_l^2 - 1) + j0$ and radius $A_l/(A_l^2 - 1)$, where $A_l = |e^{s_l\tau_n}|$. Thus $\lim_{l\to\infty} \mathrm{Re}(1/(e^{s_l\tau_n} - 1)) \geq -1/(e^{\sigma\tau_n} + 1)$, hence

$$\lim_{l\to\infty} \mathrm{Re}\left(c + \sum_{n=1}^{N} \frac{x_n}{e^{s_l\tau_n} - 1}\right) \geq c - \sum_{n=1}^{N} \frac{x_n}{e^{\sigma\tau_n} + 1} = c - \sum_{n=1}^{N} x_n + \sum_{n=1}^{N} \frac{x_n e^{\tau_n\sigma}}{e^{\tau_n\sigma} + 1}$$
$$\geq \sum_{n=1}^{N} \frac{x_n}{1 + e^{-\tau_n\sigma}} \geq \sum_{n=1}^{N} \frac{x_n}{2} > 0. \quad (4.18)$$

Multiplying (4.17) and (4.18) gives $\lim_{l\to\infty} |1/G_{bf_i}(s_l)| > 0$.

Case 3, $\sigma = 0$: Note that

$$\mathrm{Re}(1/(e^{j\omega_l\tau_n} - 1)) = -1/2,$$

so

$$\lim_{l\to\infty} \mathrm{Re}\left(c + \sum_{n=1}^{N} \frac{x_n}{e^{(\sigma_l + j\omega_l)\tau_n} - 1}\right) = c - \sum_{n=1}^{N} \frac{x_n}{2} \geq c - \frac{c}{2} > 0.$$

Thus $\lim_{l\to\infty} |1/G_{bf_i}(s_l)| \neq 0$ except possibly when the first factor of (4.15) approaches zero, i.e., $1 - e^{-s_l\tau_i} \to 0$, which occurs when $\omega\tau_i = 2\pi m$, $m \in \mathbb{Z}$.

Let

$$\mathbf{I}_n = \begin{cases} 1 & \text{if } m\tau_n/\tau_i \in \mathbb{Z}, \\ 0 & \text{otherwise.} \end{cases}$$

Now

$$\lim_{s\to j2\pi m/\tau_i} |1/G_{bf_i}(s)| = \lim_{s\to j2\pi m/\tau_i} \left| c(1 - e^{-s\tau_i}) + x_i + \sum_{\substack{n=1 \\ n\neq i}}^{N} x_n e^{-s\tau_n} \frac{1 - e^{-s\tau_i}}{1 - e^{-s\tau_n}} \right|$$

$$= \left| 0 + x_i + \lim_{s\to j2\pi m/\tau_i} \sum_{\substack{n=1 \\ n\neq i}}^{N} x_n e^{-s\tau_n} \frac{1 - e^{-s\tau_i}}{1 - e^{-s\tau_n}} \right| = x_i + \sum_{\substack{n=1 \\ n\neq i}}^{N} x_n \frac{\tau_i}{\tau_n} \mathbf{I}_n > 0,$$

where the case $\mathbf{I}_n = 1$ in the last step follows by L'Hôpital's rule. Thus

$$\lim_{l\to\infty} |1/G_{bf_i}(s_l)| > 0$$

for all sequences s_l in $\bar{\mathbb{C}}^+$ for which the limit exists, whence (4.16) holds, and thus (c).

Furthermore, since $1/G_{bf_i}(s) \neq 0$, $G_{bf_i}(s)$ is also non-singular in $\bar{\mathbb{C}}^+$, and therefore analytic as its components are analytic. (Constants as well as the exponential function are entire, i.e., analytic in $\mathbb{C}$; also note that sums, differences, and products of analytic functions are analytic; quotients of analytic functions are analytic except where the denominator equals zero.) This establishes (a). Condition (b) holds since $G_{bf_i}(s)$ is analytic in $\bar{\mathbb{C}}^+$. □

That the system is stable from input to output (input-output stable) does not guarantee that the system is stable from input to internal states (internally stable). We will next see that for certain network configurations the queue may actually be settled in equilibrium at the same time as individual sources' rates are oscillating.

4.2.3 Uniqueness of rates

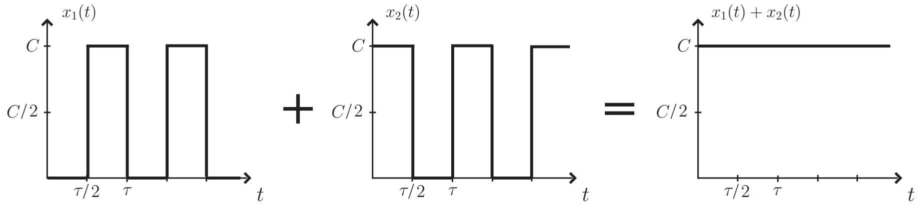
The results presented until now hold for any $x(t)$ satisfying (3.8), leaving open the question of uniqueness. It is possible for the flight sizes not to define unique rates. This is illustrated in the following example.

Example 4.2.1. *Consider a network in which two flows with equal RTTs τ share a bottleneck link of capacity C, and each has flight size $C\tau/2$. If the flows alternate between sending at rate C for time $\tau/2$ and sending at rate 0 for $\tau/2$, and if the "on" periods of flow 1 coincide exactly with the "off" periods of flow 2, then the total rate flowing into the bottleneck link is constant C and (3.8) is satisfied. This is illustrated in the graph in Figure 4.4a. However, (3.8) holds also if both sources send constantly at rate $C/2$ as in Figure 4.4b.*

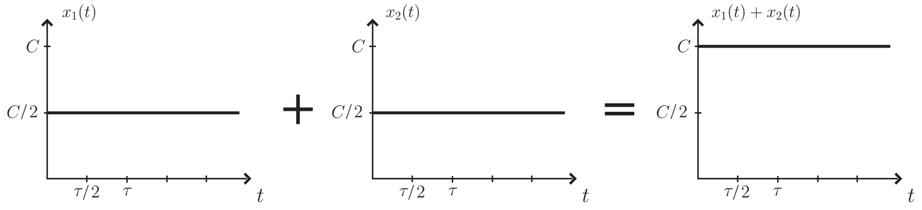
Note that (3.8) holds for any scenario similar to that described in Example 4.2.1 but with on-off periods of $\tau/2^k, k \in \mathbb{Z}^+$, and that it thus exist infinitely many solutions $(b, x(t), f)$ which corresponds to different burstiness patterns.

Now let us consider the case of small perturbations in variables. Note that sustained oscillations in the rates for constant flight sizes correspond to marginally stable (single pure imaginary) poles in the transfer function $G_{xf}(s)$ describing the dynamics from flight sizes to rates. Let z_k be the kth element of the vector z, then define $\text{diag}(z_k)$ as a square matrix with the elements of z on the diagonal. Also let $\mathbb{I}$ be a column vector with N elements which all are equal to 1. Taking the Laplace transform of (4.13) and eliminating ΔB, gives

$$\text{diag}(se^{s\tau_k})\Delta F(s) = \left(\frac{1}{c}x\mathbb{I}^{\mathrm{T}} + \text{diag}(e^{s\tau_k} - 1)\right)\Delta X(s).$$



(a) Case 1.



(b) Case 2.

Figure 4.4: Illustration of non-uniqueness of rates.

So the relationship between ΔF and ΔX is given by the transfer function matrix

$$G_{xf}(s) = \left(\frac{1}{c}x\mathbb{I}^{\mathrm{T}} + \operatorname{diag}(e^{s\tau_k} - 1)\right)^{-1} \operatorname{diag}(se^{s\tau_k}).$$

The zeros of $G_{xf}(s)$ are the values of z_i at which $G_{xf}(z_i)$ loses rank, see (Skogestad and Postlethwaite, 2005) for a formal definition. Since $G_{xf}(s)$ is a square matrix its zeros are given by the poles of the inverse dynamics

$$G_{xf}^{-1}(s) = \operatorname{diag}(1/(se^{s\tau_k}))\left(\frac{1}{c}x\mathbb{I}^{\mathrm{T}} + \operatorname{diag}(e^{s\tau_k} - 1)\right).$$

Note that $se^{s\tau_k} \neq 0$ for $s \neq 0$ and thus $\operatorname{diag}(1/(se_{s\tau_i}))$ is never singular outside the origin. This implies that there are no purely imaginary poles of $G_{xf}^{-1}(s)$ and hence no purely imaginary zeros of $G_{xf}(s)$. Therefore we know that we can not have any pole-zero cancellations on the positive and negative part of the imaginary axis for $G_{xf}(s)$. Recall from linear algebra that for matrices A and B,

$$\operatorname{Rank} AB \leq \min\{\operatorname{Rank} A, \operatorname{Rank} B\}.$$

Thus $G_{xf}^{-1}(s)$ loses rank whenever

$$\frac{1}{c}x\mathbb{I}^{\mathrm{T}} + \operatorname{diag}(e^{s\tau_k} - 1)$$

does. This occurs whenever $e^{s\tau_k} = e^{s\tau_i} = 1$ for some $i \neq k, i = 1, \ldots, N, k = 1, \ldots, N$, which is the case for any s such that $s\tau_i = j2\pi b$ and $s\tau_k = j2\pi a$ holds simultaneously for integers a and b. From this we can conclude that $G_{xf}(s)$ has poles on the imaginary axis when one flow has a RTT which is a rational multiple of another flow's RTT.

Let us study the case of $N = 2$ flows with equal equilibrium rates $x_1 = x_2 = C/2$ and RTTs such that $\tau_1/\tau_2 = b/a$ for integers a and b. The linearized rates are given by

$$\begin{pmatrix} \Delta X_1(s) \\ \Delta X_2(s) \end{pmatrix} = G_{xf}(s) \begin{pmatrix} \Delta F_1(s) \\ \Delta F_2(s) \end{pmatrix} = \begin{pmatrix} \frac{s\left(2-e^{-s\tau_2}\right)}{2-e^{-s\tau_1}-e^{-s\tau_2}} & \frac{-se^{-s\tau_1}}{2-e^{-s\tau_1}-e^{-s\tau_2}} \\ \frac{-se^{-s\tau_2}}{2-e^{-s\tau_1}-e^{-s\tau_2}} & \frac{s\left(2-e^{-s\tau_1}\right)}{2-e^{-s\tau_1}-e^{-s\tau_2}} \end{pmatrix} \begin{pmatrix} \Delta F_1(s) \\ \Delta F_2(s) \end{pmatrix}$$

which we know will oscillate due to the poles in

$$s_k = j2\pi \cdot k \cdot \min\{b/\tau_1, a/\tau_2\}, \qquad k = \pm 1, 2, \dots,$$

on the imaginary axis. Consider the transfer function from the flight sizes to the total rate $\Delta Y = \Delta X_1 + \Delta X_2$ flowing into the link, it is given by

$$G_{yf}(s) = \begin{pmatrix} 1 & 1 \end{pmatrix} G_{xf}(s) = \begin{pmatrix} \frac{2s\left(1-e^{-s\tau_2}\right)}{2-e^{-s\tau_1}-e^{-s\tau_2}} & \frac{2s\left(1-e^{-s\tau_1}\right)}{2-e^{-s\tau_1}-e^{-s\tau_2}} \end{pmatrix}.$$

We have by L'Hôpital's rule the limit

$$\lim_{s \to s_k} G_{yf}(s) = \lim_{s \to s_k} \begin{pmatrix} \frac{2\left(1-e^{-s\tau_2}\right)+2s\tau_2 e^{-s\tau_2}}{\tau_1 e^{-s\tau_1}+\tau_2 e^{-s\tau_2}} & \frac{2\left(1-e^{-s\tau_1}\right)+2s\tau_1 e^{-s\tau_1}}{\tau_1 e^{-s\tau_1}+\tau_2 e^{-s\tau_2}} \end{pmatrix}$$
$$= \begin{pmatrix} \frac{2\tau_2 s_k}{\tau_1+\tau_2} & \frac{2\tau_1 s_k}{\tau_1+\tau_2} \end{pmatrix}.$$

From the existence of this limit we can conclude that the purely imaginary s_k are not poles of $G_{yf}(s)$. This highlights the fact that sustained oscillations in the individual rates due to the marginally stable poles still maintain a constant rate flowing into the bottleneck link. It corresponds to that the marginally stable poles appearing in $G_{xf}(s)$ are canceled by zeros introduced in the summation of the rates when creating $G_{yf}(s)$. Conversely, for any periodic function $x_1(t)$ with period τ_1/b, perturbations about the mean with $x_2(t) = -x_1(t)$ will satisfy (4.13). The example that follows illustrates this.

Example 4.2.2. *Consider two window based flows sharing a bottleneck link with capacity $c = 100$ Mbit/s, with 1040 byte packets, and an equilibrium queuing delay of $b = 8.16\,ms$. First, a scenario with $f_1 = 650$ packets, $f_2 = 2f_1$, $d_1 = 100\,ms$, $d_2 = 208.16\,ms$ is simulated in NS-2. For this case $\tau_2 = 2\tau_1 = 216.32\,ms$. This rational ratio $a = 2, b = 1$ suggests sustained oscillations at frequencies $kb/\tau_1 \approx (9.25k)$ Hz where $k \in \mathbb{Z}^+$. The upper left plot in Figure 4.5 shows the single-sided amplitude spectrum (computed by the FFT) of the sending rate of source 1, sampled every 5 ms. The spikes in the plot agree with our prediction. The upper right hand plot of the amplitude spectrum of the queue size lacks such sustained oscillations; it is stable in line with Theorem 4.2.2. (While the individual sending rates oscillate, their sum does not, yielding a stable link buffer.)*

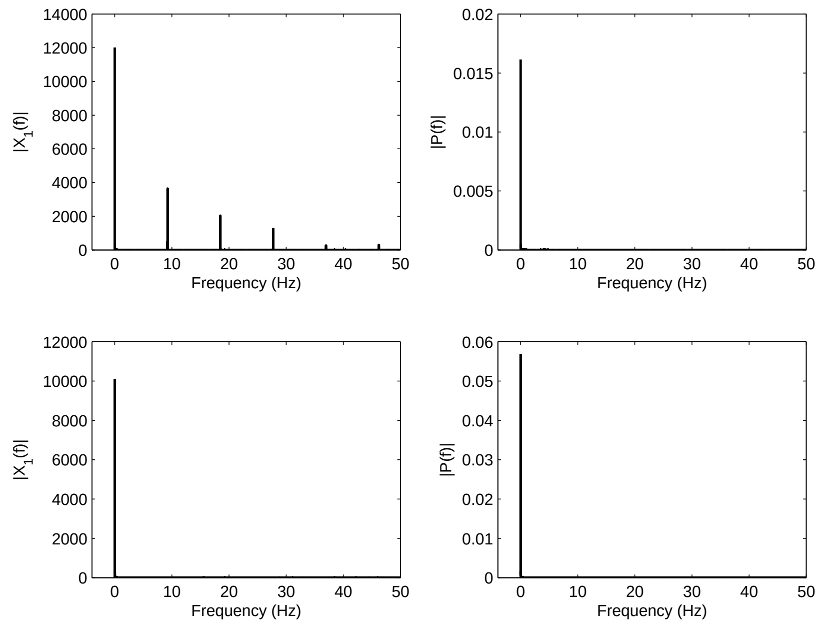


Figure 4.5: Single-sided amplitude spectrum for the rate of source 1 (left) and bottleneck queue (right). Upper plots: $a = 2, b = 1$. Lower plots: $a = 2079, b = 1352$.

The two lower plots are for a similar scenario, with instead $d_2 = 158.16\,ms$. *Since* $\tau_1/\tau_2 = b/a = 1352/2079$*, the sustained oscillations will be at* $kb/\tau_1 = (12500k)\,Hz$*. Accordingly, the amplitude spectrum up to 50 Hz of the source rate lacks spikes except at the zero mode. Again the queue is non-oscillatory.*

In summary, as it seems window based sources' rates are unique (and linearly stable) unless one flow has a RTT which is a rational multiple of another flow's RTT.

4.2.4 Model simplifications

Though the ACK-clocking model (3.10) is very accurate, as will be shown in Chapter 5, it may due to its complexity not be tractable for many applications. We will here provide some naive guidelines for how the model can be simplified for analytical purposes. In an analysis this should be done to match the accuracy requirements of the application considered. To keep the discussion as simple as possible we will focus on the single link case (3.8).

Studying (3.8) we observe that the integration (3.8a) is linear in the states why the complexity in the model is, mainly, due to the N constraints (3.8b) defining the rates $x_n(t)$. We will, thus, be concerned with approximating the integral equation (3.8b). Let

$$H_n(t,z) = \int_t^z x_n(s)ds - f_n(z), \tag{4.19}$$

where, by (3.8b),

$$H_n(t, t+\tau_n(t)) = 0. \tag{4.20}$$

Intuitively, better approximations of the constraints (3.8b) should improve overall model accuracy. However, due to the coupling between the constraints (3.8b) and the integration (3.8a), even though we are able to quantify the accuracy of the approximation of $H_n(t,z)$, more rigorous analysis than provided here is needed to quantify the accuracy in the output queuing delay b. The discussion that follows is, thus, rather informal and results are qualitative. See also the discussion on approximations based on numerical quadrature techniques in Appendix A.

By Taylor expansion

Under the assumption that $x_n(t)$ and $f_n(t)$, and subsequently $H_n(t,z)$, are sufficiently smooth and thus analytic, the integral $H_n(t, t+\tau_n(t))$ can be Taylor expanded around t. For sufficiently small round trip times approximate ODE models can then be obtained from different orders of truncation of this series expansion. We have that

$$0 = H_n(t, t+\tau(t)) = -f_n(t) + \sum_{k=1}^{\infty} \left(\left. \frac{d^{k-1}x_n(\xi)}{d\xi^{k-1}} \right|_{\xi=t} - \left. \frac{d^k f_n(\xi)}{d\xi^k} \right|_{\xi=t} \right) \frac{\tau_n^k(t)}{k!}.$$

Hence

$$0 = H_n(t, t+\tau(t)) = -f_n(t) + (x_n(t) - \dot{f}_n(t))\tau_n(t) + \mathcal{O}(\tau_n^2),$$

and thus a second order truncation and dividing by $\tau_n(t)$ gives the $\mathcal{O}(\tau)$ rate approximation

$$x_n(t) \approx \frac{f_n(t)}{\tau_n(t)} + \dot{f}_n(t). \tag{4.21}$$

Similarly, by third and fourth order truncations we have, after variable transformations $\dot{y}_n = x_n$ and $z_n = y_n - f_n$, the models

$$\frac{\tau_n(t)^2 \ddot{z}_n(t)}{2} + \tau_n(t)\dot{z}_n(t) - f_n(t) \approx 0 \tag{4.22}$$

and

$$\frac{\tau_n(t)^3 \dddot{z}_n(t)}{6} + \frac{\tau_n(t)^2 \ddot{z}_n(t)}{2} + \tau_n(t)\dot{z}_n(t) - f_n(t) \approx 0 \tag{4.23}$$

respectively. The trade-off between model tractability and accuracy now lies in the complexity in the ODE. Let us compare these simplifications in a simulation experiment.

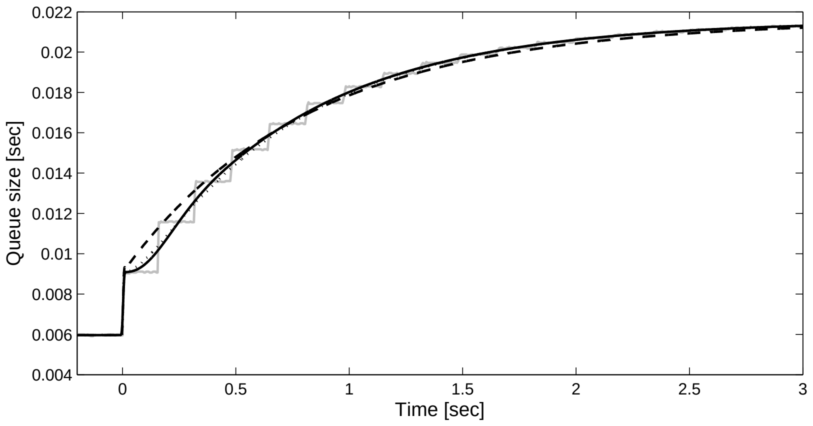


Figure 4.6: Validation experiment. Solid grey line: NS-2 simulation. Dashed black line: first order rate model (4.21) (same as (4.29)). Dotted black line: second order rate model (4.22). Solid black line: third order rate model (4.23).

Example 4.2.3. *Consider a window based source sharing a single bottleneck link of capacity $c = 200$ Mbit/s with (UDP) cross traffic with constant rate $x_c = 160$ Mbit/s. The window based source has a round trip propagation delay of $d = 150$ ms and its flight size is initially set to a constant size $f_0 = 750$ packets; packet size is set to $\rho = 1040$ bytes. At time $t = 0$ s the system is perturbed from equilibrium by increasing the flight size of the window based source with 75 packets to $f(t) = 825$ packets. The solid grey line in Figure 4.6 is the queue size when this scenario is simulated in NS-2. The dashed black line corresponds to the model based on the first order rate model (4.21), the dotted black line to the model based on the second order rate model (4.22) and the solid black line to the model based on the third order rate model (4.23).*

As expected we observe in this example that the model using the simplest rate approximation (4.21) is less accurate than the other two which it is compared with. However, we see that for this particular scenario there is no fundamental difference between the two higher order approximations (studying the Euclidean length of the residuals reveal that the highest order approximation actually is slightly more accurate here).

By Padé approximations

If the objective is to derive approximative models in the form of ODEs, an alternative to the previous Taylor expansion procedure may be to study the linearized model in the Laplace domain (4.14), and apply, for example, suitable orders of Padé approximations (Baker and Graves-Morris, 1996) to approximate the exponential function $e^{-s\tau_n}$ originating from to the physical delays in the system. A corresponding nonlinear ODE can then be "reverse engineered" from the linear model to approximate the original nonlinear delay DAE model. A Padé approximant of a given order (m, n) is the "best" approximation of a function by a rational function. The integers m refers to the order of the numerator polynomial and the integer n to the order of the denominator polynomial.

If we use a $(1, 0)$ Padé approximation, that is $e^{-s\tau} \approx 1 - s\tau$, the transfer function from the flight size of the ith source to the queue (4.15) becomes

$$G_{bf_i}(s) \approx G_{bf_i}^{1,0}(s) = \frac{1/\tau_i}{sx_c + \sum_{n=1}^{N} x_n/\tau_n}. \tag{4.24}$$

Similarly, if we denote the transfer function from the cross traffic x_c to buffer size b with $G_{bx_c}(s)$, see (4.14), we have

$$G_{bx_c}(s) \approx G_{bx_c}^{1,0}(s) = \frac{1}{sx_c + \sum_{n=1}^{N} x_n/\tau_n}. \tag{4.25}$$

Using (4.24) and (4.25) we get

$$\begin{aligned}\Delta B(s) &= \sum_{n=1}^{N} G_{bf_n}(s)\Delta F_n(s) + G_{bx_c}(s)\Delta X_{bx_c}(s) \\ &\approx \frac{1}{sx_c + \sum_{n=1}^{N} x_n/\tau_n}\left(\sum_{n=1}^{N} \frac{\Delta F(s)}{\tau_n} + \Delta X_{bx_c}(s)\right). \end{aligned}\tag{4.26}$$

Hence, by applying the inverse Laplace transform to (4.26), the approximative linear time-domain ODE

$$\delta\dot{b}(t) = -\frac{1}{x_c}\sum_{n=1}^{N}\frac{x_n}{\tau_n}\delta b(t) + \frac{1}{x_c}\sum_{n=1}^{N}\frac{1}{\tau_n}\delta f_n(t) + \frac{1}{x_c}\delta x_c(t) \tag{4.27}$$

results.

Let us study a specific nonlinear law with linear dynamics (4.27). It is trivial to verify that (4.27) is obtained when linearizing the nonlinear ODE

$$\dot{b}(t) = \frac{1}{x_c(t)}\left(\sum_{n=1}^{N}\frac{f_n(t)}{d_n + b(t)} + x_c(t) - c\right) \tag{4.28}$$

around the equilibrium (b, f, x_c). Note that in the case of no cross traffic, that is, $x_c = 0$, this ODE reduces to an algebraic equation.

Using a $(0, 1)$ Padé approximation, that is, $e^{-s\tau} \approx 1/(1 + s\tau)$, gives in the same manner as for the $(1, 0)$ case the model

$$\dot{b}(t) = \frac{1}{c}\left(\sum_{n=1}^{N}\left(\frac{f_n(t)}{d_n + b(t)} + \dot{f}_n(t)\right) + x_c(t) - c\right). \tag{4.29}$$

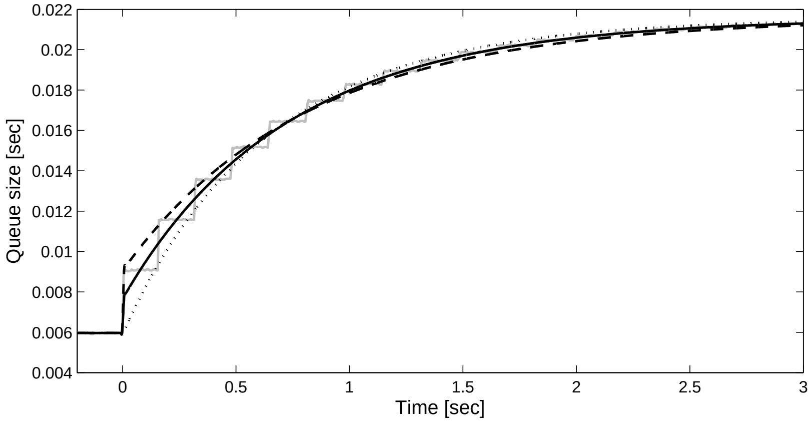


Figure 4.7: Validation experiment. Solid grey line: NS-2 simulation. Dotted black line: $(1, 0)$ model (4.28). Dashed black line: $(0, 1)$ model (4.29). Solid black line: $(1, 1)$ model (4.30).

A more accurate $(1, 1)$ approximation $e^{-s\tau} \approx (1 - s\tau/2)/(1 + s\tau/2)$ yields, according to the same procedure, the more complex model

$$\dot{b}(t) = \frac{2}{x_c(t) + c} \left(\sum_{n=1}^{N} \left(\frac{f_n(t)}{d_n + b(t)} + \frac{\dot{f}_n(t)}{2} \right) + x_c(t) - c \right). \tag{4.30}$$

We will now compare these models.

Example 4.2.4. *The scenario is identical to the one descibed in Example 4.2.3. The system is simulated in NS-2 and compared with the models based on the different Padé approximations explictly dicussed. The solid grey line in Figure 4.7 is the queue size when the system is simulated in NS-2, the dotted black line the model (4.28) based on the Padé* $(1, 0)$ *approximation (identical to the model based on (4.21)), the dashed black line the model (4.29) which corresponds to Padé* $(0, 1)$ *approximation, the solid black line is the model (4.30) which is derived using a Padé* $(1, 1)$ *approximation.*

It is clear from the example that the model based on the more accurate approximation $(1, 1)$, tracking the average of the queue during the whole transient, is superior to the more coarse approximations which seem to suffer from more phase shift. To better capture the stepwise changes in the queue, we need to make even higher order approximations of the exponential function, and consequently higher order ODE models.

All of the approximate ODE models previously discussed in this section are based on small τ approximations. However, $\tau(t)$ need not be small, in particular

$\tau(t)$ does not approach zero in the fluid limit of many packets. Thus, (3.8) should be used whenever it results in a tractable problem formulation.

4.3 Relation between flight size and window size

We have discussed in Section 3.4.3 that there may be a mismatch between the window size and the flight size in the case of an abrubt window reduction. More specifically, due to that the endpoint control protocol cannot immediately withdraw packets from the network, the flight size cannot be decreased faster than the arrival rate of the receiced ACKs. We will here investigate the relation between the flight size f_n and the congestion window w_n in more detail. Subscripts n will be dropped for the ease of notation.

Recall from (3.3) that the flight size f is defined from the equality

$$\int_t^{t+d+b(t)} x(s)\,ds = f(t+d+b(t)), \quad t \geq 0,$$

where x is the instantaneous sending rate, b the queuing delay, and d the propagation delay. Let $0 < t_1 < t_2 < \dots,$ be the time instances when an ACK arrives at the sender. Denote the sequence of consecutive instances a round trip time apart as $0 < \mathbf{t}_1 < \mathbf{t}_2 < \dots,$ and suppose that $\mathbf{t}_1 = t_1$, i.e., the first packet was sent at $t = 0$. Note that the set of round trip epochs $\{\mathbf{t}_\ell\}$ is a subset of the ACK times $\{t_k\}$.

Given the congestion window w, the corresponding sending rate x is not uniquely defined. The congestion window only determines what the *average* rate over a round trip time should be. It is common that if w is increased by a certain number of packets these packets are instantaneously put on the network, while if w is decreased the packet decrease on the network depends on when the next few ACKs arrive to the sender. Many other implementations of the congestion window changes are possible, e.g., smoothing a window increase over a certain time interval. Obviously, the relation between the actual sending rate and the congestion window depends on the protocol implementation. Next, we derive some fundamental bounds on the difference between the congestion window w and the flight size f.

In general, it is hard to obtain a better bound on the point-wise difference between w and f than

$$\sup_{t\geq 0} |w(t) - f(t)| \leq \sup_{t\geq 0} w(t) = w_{\max},$$

where $w_{\max}$ denotes the largest congestion window over a session. Note that the inequality follows simply from the previous argument: if the congestion window w is decreased, then the flight size f is not instantaneously decreased but decreases only as the next ACKs arrive to the sender.

Let us now consider the L_1 norm of the difference between w and f:

$$\|w-f\|_{L_1} = \sup_{T>0} \frac{1}{T}\int_0^T |w(t)-f(t)|dt.$$

It is a measure of how close w and f are in average. Let us assume that the congestion window w is updated only at the round trip times $\mathbf{t}_\ell$, $\ell = 1, 2, \ldots$, so that $w(t) = f(\mathbf{t}_\ell)$, $\mathbf{t}_\ell \le t < \mathbf{t}_{\ell+1}$, i.e., $w(t) = f(\mathbf{t}_{\ell(t)})$ where $\ell(t) = \ell$, $\mathbf{t}_\ell \le t < \mathbf{t}_{\ell+1}$. Note that for $t \in (\mathbf{t}_\ell, \mathbf{t}_{\ell+1})$, it holds by the mean value theorem that $f(t) = f(\mathbf{t}_\ell) + (t-\mathbf{t}_\ell)f'(\xi)$ for some $\xi \in (\mathbf{t}_\ell, \mathbf{t}_{\ell+1})$ under the assumption that f is differentiable. Hence, $|f(\mathbf{t}_\ell) - f(t)| \le (\mathbf{t}_{\ell+1} - \mathbf{t}_\ell)|f'(\xi)|$. With $f'_{\max} = \sup_{\ell=1,2,\ldots} |f'(\xi(\ell))|$ and $\mathrm{RTT}_{\max} = \sup_{\ell=1,2,\ldots}(\mathbf{t}_{\ell+1} - \mathbf{t}_\ell)$, we thus have $|f(\mathbf{t}_\ell) - f(t)| \le \mathrm{RTT}_{\max} f'_{\max}$. Then,

$$\begin{aligned}\|w-f\|_{L_1} &= \sup_{T>0} \frac{1}{T}\int_0^T |f(\mathbf{t}_{\ell(t)}) - f(t)|dt \\ &\le \sup_{T>0} \frac{1}{T}\int_0^T \mathrm{RTT}_{\max} f'_{\max} dt \\ &\le \mathrm{RTT}_{\max} f'_{\max}.\end{aligned}$$

The approximation error between w and f is thus of the order of the round trip time.

To summarize, we have proven the following theorem.

Theorem 4.3.1. *The following relations between the congestion window w and the flight size f hold: if $w(t)$ is updated at $t \in \{\mathbf{t}_\ell\}_{\ell=1,2,\ldots}$, then $w(\mathbf{t}_\ell) = f(\mathbf{t}_\ell)$ and $\|w-f\|_{L_1} \le \mathrm{RTT}_{\max} f'_{\max}$.*

4.4 Stability of FAST TCP

In this this part we will analyze the stability of FAST TCP. We will consider the case of an arbitrary number of sources and a single bottleneck. The analysis itself highlights the power of the fluid flow abstraction as well as how important it is that the models used are tailored for their purpose and that the final result is properly evaluated.

Before proceeding with the analysis, recall the model associated with the protocol and which were derived in Section 3.6. The dynamics of the protocol is modeled according to Figure 3.20 with window and estimator dynamics given by (3.22) and (3.23) respectively, i.e.,

$$\begin{aligned}\dot{w}(t) &= \frac{\log(1-\gamma\xi(t))}{h_k(t)}\left(w(t) - \frac{\alpha}{\xi(t)}\right), \\ \dot{\hat{q}}(t) &= \frac{\log(1-\sigma(t))}{h'_{k'}}\left(\hat{q}(t) - u_q(t)\right),\end{aligned}$$

where $\xi = u_{\hat{q}}/(d + u_{\hat{q}})$ and $\sigma(t) = \min\{\kappa/u_w(t), \nu\}$.

4.4.1 System equilibrium

In this part we investigate the uniqueness of the system equilibrium under the assumption that the routing matrix R has full rank. This is easily done using the network utility maximization framework introduced in Section 4.1.1.

The equilibrium condition of the FAST TCP window control is given by letting $\dot{w} = 0$ in (3.22), it is

$$w = Q(q) = \frac{\alpha(d+q)}{q}.$$

Subsequently, the corresponding equilibrium rate is

$$x = F_n(q) = \frac{w}{d+q} = \frac{Q(q)}{d+q} = \frac{\alpha}{q}.$$

Since $F_n(q)$ is monotonically decreasing for $\{q \geq 0\}$ its inverse exist, and it is monotonically decreasing as well. This implies, according to the discussion in Section 4.1.1, a strictly concave utility function and thus unique rates x, aggregated queuing delays q and individual queuing delays b in equilibrium. Furthermore since $Q(q)$ is monotone equilibrium windows w are unique as well.

In particular we have that

$$q = f(x) = \frac{\alpha}{x}.$$

The FAST TCP utility function is achieved by integrating this expression with respect to x,

$$U(x) = \alpha \log(x).$$

From the discussion on the link between protocol utility functions and fairness properties in Section 2.5.3, it is evident that FAST TCP is $(\alpha_n, 1)$-proportionally fair.

The dynamical properties around the unique equilibrium (w, q) is investigated next.

4.4.2 Linear model

Linearized models for the different sub-parts of the system will now be derived. The final result is an open loop transfer function that subsequently will be used to evaluate linear stability of the system around the equilibrium point. Without loss of generality we assume no forward propagation delay.

Protocol

We start with deriving a linear model for the FAST TCP window control and estimator dynamics. We will be concerned with deriving transfer functions for each block in Figure 4.8, approximating the protocol part of the system in Figure 3.20 under FAST TCP.

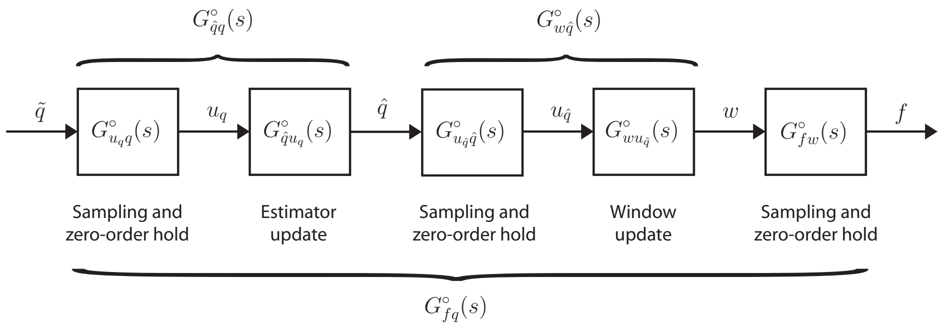

$G^{\circ}_{\hat{q}q}(s)$
$G^{\circ}_{w\hat{q}}(s)$
$\tilde{q}$
$G^{\circ}_{u_q q}(s)$
u_q
$G^{\circ}_{\hat{q}u_q}(s)$
$\hat{q}$
$G^{\circ}_{u_{\hat{q}}\hat{q}}(s)$
$u_{\hat{q}}$
$G^{\circ}_{wu_{\hat{q}}}(s)$
w
$G^{\circ}_{fw}(s)$
f
Sampling and zero-order hold
Estimator update
Sampling and zero-order hold
Window update
Sampling and zero-order hold
$G^{\circ}_{fq}(s)$

Figure 4.8: Linear model of the FAST TCP protocol dynamics.

Window control Before proceeding with the actual linearization suitable approximations of the sampling rate must be found. In FAST TCP the window control mechanism is updated once per received window (once per RTT), see Section 3.6. Thus, for the window control we will use $h_k(t) \approx d + u_{\hat{q}}(t)$. Under this approximation the window dynamics (3.22) becomes

$$\dot{w}(t) = \frac{\log\left(1 - \gamma u_{\hat{q}}(t)/(d + u_{\hat{q}}(t))\right)}{d + u_{\hat{q}}(t)} \left(w(t) - \frac{\alpha(d + u_{\hat{q}}(t))}{u_{\hat{q}}(t)} \right). \tag{4.31}$$

If we assume that the propagation delay d is estimated accurately, the system can be linearized around the equilibrium point (w, q). Note that in equilibrium $\hat{q} = \tilde{q} = q = u_q = u_{\hat{q}}$ and $w = u_w$. Furthermore, note that due to the equilibrium properties of FAST TCP, see (3.18), we have

$$\frac{\alpha}{q} = \frac{w}{d+q} = \frac{(c - x_c)\alpha_n}{\sum_m \alpha_m}.$$

Linearization of (4.31) around the equilibrium point yields

$$\delta\dot{w}(t) = \frac{\log\left(1 - \gamma q/\tau\right)}{\tau} \left(\delta w(t) + \frac{\alpha d}{q^2} \delta u_{\hat{q}}(t) \right).$$

The corresponding transfer functions from $u_{\hat{q}}$ to w is

$$G^{\circ}_{wu_{\hat{q}}}(s) = \frac{\alpha d/q^2}{\zeta s - 1}$$

where

$$\zeta = \left(\frac{\log\left(1 - \gamma q/\tau\right)}{\tau} \right)^{-1}.$$

It remains to determine the transfer function from $\hat{q}$ to $u_{\hat{q}}$, denoted $G^{\circ}_{u_{\hat{q}}\hat{q}}(s)$, modeling the effect of sampling (3.11),

$$g[k] = \int_0^\infty \delta(s - t_k) g(s)\, ds, \qquad k = 0, 1, 2, \dots,$$

and the zero-order hold (3.12),

$$g(t) = g(t_k), \qquad t_k \leq t < t_{k+1},$$

in series, see Figure 3.20. Note that in FAST TCP the window control sampling rate h_k is a multiple of the estimator sampling rate $h'_{k'}$, see (3.21), and thus only the zero-order hold operating on per RTT basis is accounted for in the model.

In equilibrium the RTT is constant and thus the window control sample interval $h_k(t)$ is so as well. To be able to analyze the system in the frequency domain we will assume that the effect of the time-varying sampling is negligible at relevant frequencies. This assumption is supported by that the dynamics of this phenomenon is fast and that the rest of the system is of low-pass character. In addition, the presence of the sampler implies that the system is not time-invariant, even under the assumption of constant sample intervals (Åström and Wittenmark, 1997). Strictly, this means that the system dynamics can not be characterized by ordinary transfer functions. However, we will, supported by the low-pass character of the system, assume that the aliasing that occur is not severe and that distortion effects are neglectable. This manifest in the frequency domain as approximating the transfer function of the sampler (3.11) as unity.

The Laplace transform of the zero-order hold (3.12) with constant sample intervals h is

$$\frac{1 - e^{-sh}}{sh}. \tag{4.32}$$

Thus, since $h_k = d + q = \tau$, we have that

$$G^{\circ}_{u_{\hat{q}}\hat{q}}(s) = \frac{1 - e^{-s\tau}}{s\tau}.$$

In summary, the window control transfer function of a source becomes

$$G^{\circ}_{w\hat{q}}(s) = G^{\circ}_{wu_{\hat{q}}}(s) G^{\circ}_{u_{\hat{q}}\hat{q}}(s) = \frac{\alpha d/q^2}{\zeta s - 1} \cdot \frac{1 - e^{-s\tau}}{s\tau}, \tag{4.33}$$

see Figure 4.8.

Estimator dynamics We will now focus on the estimator part of the FAST TCP protocol. We start with approximating the sample time used by the FAST TCP estimator algorithm before linearizing the dynamics.

The queuing delay estimator in FAST TCP is updated at every ACK arrival. An approximation $h'_{k'}(t) \approx (d + u_q(t))/u_w(t)$ corresponding to the average ACK inter-arrival time over a RTT is thus chosen as sample rate approximation.

Consider the case when window size $w(t)$ and parameters κ and ν satisfies $\kappa/w(t) \leq \nu$. This condition is typically fulfilled under normal parameter values κ and ν due do large window sizes in high-bandwidth product networks (recall from Section 3.6 that the default setting is $\kappa = 3$ and $\nu = 1/4$ which implies that the window $w(t)$ only needs to be larger than 12 packets for the condition to be satisfied). Subsequently, we have from (3.23) that

$$\dot{\hat{q}}(t) = \frac{u_w(t)\log\left(1 - \kappa/u_w(t)\right))}{d + u_q(t)}\left(\hat{q}(t) - u_q(t)\right). \tag{4.34}$$

Linearization of (4.34) around the equilibrium (w, q) now gives

$$\delta\dot{\hat{q}}(t) = \frac{w\log\left(1 - \kappa/w\right))}{\tau}\left(\delta\hat{q}(t) - \delta u_q(t)\right), \tag{4.35}$$

(note that $\delta\dot{\hat{q}}(t)$ is independent of the feedback $\delta u_w(t)$). Applying the Laplace transform to (4.35) gives the transfer function from u_q to $\hat{q}$,

$$G^{\circ}_{\hat{q}u_q}(s) = -\frac{1}{\eta s - 1}, \tag{4.36}$$

with

$$\eta = \left(\frac{w\log\left(1 - \kappa/w\right)}{\tau}\right)^{-1}.$$

Analogous to the window control case we need to account for the sampler and the zero-order hold function, i.e., the dynamics from $\tilde{q}$ to u_q. We will at this stage also make the analogous approximation that the estimator sampling time is constant $h'_{k'} = \tau/w$ and that distortion is neglectable. That it is sampled faster than the window control, per received ACK instead of per RTT, is just favorable for the approximation. It is realized from the derivation of the window update transfer function (4.33), modeling a sampler and a zero-order hold in series also, that the transfer function from $\tilde{q}$ to u_q is

$$G^{\circ}_{u_q q}(s) = \frac{1 - e^{-s\tau/w}}{s\tau/w}$$

(note the slight abuse of notation in the naming convention of $G_{u_q q}$, we have here used q instead of $\tilde{q}$).

Summarizing, the transfer function of the estimator dynamics of a source is

$$G^{\circ}_{\hat{q}q}(s) = G^{\circ}_{\hat{q}u_q}(s)G^{\circ}_{u_q q}(s) = -\frac{1}{\eta s - 1}\cdot\frac{1 - e^{-s\tau/w}}{s\tau/w}, \tag{4.37}$$

see Figure 4.8.

Flight size control In the equilibrium analysis we are concerned with small window changes. We thus assume that the window pacing and and burstiness reduction algorithms that FAST TCP applies are passive around the operating point. Hence, the flight size control dynamics is just holding the flight size over a RTT around the equilibrium and thus consists of a zero-order hold only, cf., Figure 3.17 and Figure 3.20. The transfer function linking the continuous time window $w(t)$ with the flight size $f(t)$ thus consists of the Laplace transform of a zero-order hold in conjunction with a sampler with sampling interval $h_k = \tau$, see Figure 3.20. Consequently, the transfer function from the window size w to the flight size f is modeled as

$$G^{\circ}_{fw}(s) = \frac{1-e^{-s\tau}}{s\tau}, \tag{4.38}$$

in line with previous approximations.

Combining the window update (4.33), the estimator dynamics (4.37) and the flight size control (4.38) gives that the protocol transfer function of the nth source is given by

$$\begin{aligned} G^{\circ}_{f_n q_n}(s) &= G^{\circ}_{f_n w_n}(s) G^{\circ}_{w_n \hat{q}_n}(s) G^{\circ}_{\hat{q}_n q_n}(s) \\ &= -\frac{1-e^{-s\tau_n}}{s\tau_n} \cdot \frac{\alpha_n d_n/q^2}{\zeta_n s - 1} \cdot \frac{1-e^{-s\tau_n}}{s\tau_n} \cdot \frac{1}{\eta_n s - 1} \cdot \frac{1-e^{-s\tau_n/w_n}}{s\tau_n/w_n}, \end{aligned}$$

cf., Figure 4.8. Define the corresponding matrix transfer function accordingly

$$G^{\circ}_{fq}(s) = \operatorname{diag}\left(G^{\circ}_{f_n q_n}(s)\right). \tag{4.39}$$

Transport delay

Since FAST TCP uses queuing delay as aggregate price signal we have

$$\tilde{q}_n(t + d_n + b(t)) = b(t),$$

where the time delay is due to the backward transport delay, the packets must travel through the buffer to the receiver and back. Note that this is equal to the round trip delay since we assumed zero forward propagation delay. Ignoring the time varying variable arguments, analogously to the linearization of the ACK-clocking mechanism, the dynamics is linear and thus

$$\delta\tilde{q}_n(t + \tau_n) = \delta b(t).$$

Applying the Laplace transform we get the nth backward delay transfer function

$$G^{\circ}_{q_n b}(s) = e^{-s\tau_n^b} = e^{-s\tau_n},$$

and accordingly

$$G^{\circ}_{qb}(s) = \begin{pmatrix} e^{-s\tau_1} & e^{-s\tau_2} & \cdots & e^{-s\tau_N} \end{pmatrix}^{\mathrm{T}}. \tag{4.40}$$

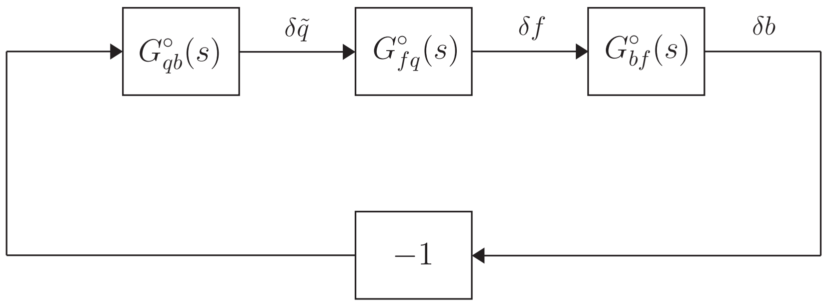


Figure 4.9: Closed loop system, FAST TCP.

Link

That FAST TCP uses queuing delay as congestion indication implies that the "link" model (describing the dynamics from flight sizes to the buffer) is simply the ACK-clocking model. Thus, from (4.14) we have

$$G^\circ_{bf}(s) = \begin{pmatrix} G^\circ_{bf_1}(s) & G^\circ_{bf_2}(s) & \cdots & G^\circ_{bf_N}(s) \end{pmatrix}, \tag{4.41}$$

where

$$G_{bf_i}(s) = \frac{1}{(1-e^{-s\tau_i})\left(c + \sum_{n=1}^N x_n \frac{e^{-s\tau_n}}{1-e^{-s\tau_n}}\right)}. \tag{4.42}$$

Loop gain

The entire closed loop system is given by the the protocol dynamics (4.39), the transport delay (4.40) and the link model (4.41), see Figure 4.9. The open loop transfer function when breaking up the loop at the buffer size δb is given by

$$L_\circ(s) = -G^\circ_{bf}(s) G^\circ_{fq}(s) G^\circ_{qb}(s) = \sum_{n=1}^N L_{\circ,n}(s) \tag{4.43}$$

where

$$L_{\circ,n}(s) = \frac{\alpha_n d_n/q^2 (1-e^{-s\tau_n})^2/(s\tau_n)^2 (1-e^{-s\tau_n/w_n})/(s\tau_n/w_n) e^{-s\tau_n}}{(\zeta_n s - 1)(\eta_n s - 1)(1-e^{-s\tau_n})\left(c + \sum_{i=1}^N x_i \frac{e^{-s\tau_i}}{1-e^{-s\tau_i}}\right)}$$

if we assume negative feedback, cf., Figure 4.2. Due to the exponential function and the sums in this expression stability analysis is non-trivial.

4.4.3 Approximating the loop gain

Due to the complexity of the loop gain (4.43) we will use a simplified version of it in the stability analysis that follows. The validity of the results using this more tractable approximation will then be investigated.

FAST TCP is primarily designed to be able to achieve high throughput and fairness in networks with high bandwidths and large latencies. In such networks sources keep a large amounts of packet in flight and propagation delays are large compared to queuing delays, this implies that $\gamma_n q/\tau_n \ll 1$ and $\kappa_n/w_n \ll \nu_n < 1$ are typical. If we consider such cases only the approximations

$$\log\left(1 - \gamma_n \frac{q}{\tau_n}\right) \approx -\gamma_n \frac{q}{\tau_n} \tag{4.44}$$

and

$$\log(1 - \frac{\kappa_n}{w_n}) \approx -\frac{\kappa_n}{w_n} \tag{4.45}$$

are valid, it follows that

$$\zeta_n \approx -\frac{{\tau_n}^2}{\gamma_n q} \tag{4.46}$$

$$\eta_n \approx -\frac{\tau_n}{\kappa_n}. \tag{4.47}$$

To further simplify the model we will approximate the exponential functions that appears in the zero-order holds and the ACK-clocking transfer functions with Padé approximations of order $(1,1)$, i.e., the approximation

$$e^{sh} \approx \frac{1 - sh/2}{1 + sh/2}. \tag{4.48}$$

Applying this to a zero-order hold gives

$$\frac{1 - e^{-sh}}{sh} \approx \frac{1}{sh/2 + 1}. \tag{4.49}$$

By applying (4.46), (4.47) and (4.49) we get an approximation of the protocol transfer function $G^\circ_{fq}(s) \approx G_{fq}(s)$,

$$G_{fq}(s) = \operatorname{diag}\left(G_{f_n q_n}(s)\right)$$

where

$$\begin{aligned} G_{f_n q_n}(s) &= -\frac{\gamma_n d_n \alpha_n/q}{(s{\tau_n}^2 + \gamma_n q)(s\tau_n/\kappa_n + 1)(s\tau_n/2 + 1)^2(s\tau_n/(2w_n) + 1)} \\ &\approx -\frac{\gamma_n(\tau_n - b)\alpha_n/b}{(s{\tau_n}^2 + \gamma_n b)(s\tau_n/\kappa_n + 1)(s\tau_n/2 + 1)^2}. \end{aligned}$$

The last approximation is due to that in the high bandwidth large latency case studied $2w_n \gg \max\{2, \kappa_n, \gamma_n b/\tau_n\}$. This implies that the zero-order hold dynamics deriving from the estimator is substantially faster than the other parts of the protocol dynamics, thus, it is negligible in our context.

Replacing the exponential functions in the model (4.42) by (4.48) yields $G^{\circ}_{bf_i}(s) \approx G_{bf_i}(s)$ with

$$G_{bf_i}(s) = \frac{2}{c + x_c} \cdot \frac{(s\tau_n/2 + 1)/\tau_n}{s + 2/(c + x_c)\sum_{n=1}^{N} x_n/\tau_n}.$$

Accordingly $G^{\circ}_{bf}(s) \approx G_{bf}(s)$ where

$$G_{bf}(s) = \begin{pmatrix} G_{bf_1}(s) & G_{bf_2}(s) & \cdots & G_{bf_N}(s) \end{pmatrix}^{\mathrm{T}}$$

analogous to (4.41).

We are now ready to form the approximative open loop transfer function that we will use as model in the analysis,

$$L(s) = -G_{bf}(s)G_{fq}(s)G^{\circ}_{qb}(s) = \lambda \sum_{n=1}^{N} \frac{\mu_n}{\tau_n} L_n(s) \tag{4.50a}$$

where

$$L_n(s) = \frac{\gamma_n(\tau_n - b)e^{-s\tau_n}}{(s{\tau_n}^2 + \gamma_n b)(s\tau_n/\kappa_n + 1)(s\tau_n/2 + 1)\,(s + \lambda/\hat{\tau})}, \tag{4.50b}$$

$$\mu_n = \frac{x_n}{c} = \frac{\alpha_n}{cb} = \frac{\alpha_n}{\sum_{i=1}^{N}\alpha_i}, \tag{4.50c}$$

$$\frac{1}{\hat{\tau}} = \sum_{n=1}^{N} \frac{\mu_n}{\tau_n}, \tag{4.50d}$$

$$\lambda = \frac{2c}{c + x_c}. \tag{4.50e}$$

Note that α_n is the number of packets a source tries to queue in the network, thus the queuing delay $b = \sum_n \alpha_n/c$ in equilibrium, and hence the last equality in (4.50c) follows.

Remark 4.4.1. *We will sometimes let $b \to 0$, it is then assumed that $\alpha_n \to 0$ with α_m/α_n fixed, such that μ_n is well defined.*

When no cross traffic is present, i.e., $x_c = 0$, we can interpret $\hat{\tau}$ as a weighted harmonic mean value of the round trip delays τ_n. In particular, when all flows have equal α_n, giving $\mu_n = 1/N$, $\hat{\tau}$ is the harmonic mean of τ_n.

4.4.4 Stability analysis

In the FAST TCP model (4.50b), the case when the queuing delay is $b = 0$ is intuitively the least stable, as increasing b reduces the gain and introduces phase lead, both of which intuitively improve stability. To formalize this, Lemma 4.4.1 and Lemma 4.4.2 will be used to place bounds on the values of the minimum of $\mathrm{Re}(F_n(j\omega))$ which is used in Lemma 4.1.2.

Lemma 4.4.1. *Consider a complex half-plane*

$$H = \{z : \operatorname{Im}((z - z^*)e^{j\beta}) \geq 0\}$$

containing 0. Consider also a function $G : \mathbb{R}^+ \mapsto \mathbb{C}$ with

$$G(\nu) = r(\nu)e^{j\theta(\nu)}$$

where $r \geq 0$ and θ are continuous decreasing real functions, r is unbounded as $\nu \to 0$, and $\beta + \theta(0) = \Phi$ with $\Phi \in (0, \pi)$ the angle between the edge of the half-plane and the tail of the spiral G. If

$$\{G(\nu) : \nu \in \mathbb{R}^+\} \subset H$$

then

$$\{\eta(\nu)e^{j\phi(\nu)}G(\nu) : \nu \in \mathbb{R}^+\} \subset H$$

for any

$$\eta : \mathbb{R}^+ \mapsto [0,1], \quad \phi : \mathbb{R}^+ \mapsto [0, \pi - \Phi).$$

In particular, if $\theta(0) = -\pi/2$ then taking $\beta = \pi/2$ gives

$$\min_\nu \operatorname{Re}(\eta(\nu)e^{j\phi(\nu)}G(\nu)) \geq \min_\nu \operatorname{Re}(G(\nu)) \tag{4.51}$$

for any

$$\eta : \mathbb{R}^+ \mapsto [0,1], \quad \phi : \mathbb{R}^+ \mapsto [0, \pi).$$

Proof. First, consider the tail of the spiral, $\nu \in [0, \bar{\nu}]$ where $\theta(0) - \theta(\bar{\nu}) = \phi$. The image of the tail (under the rotation and scaling) is entirely in the sector

$$\{z : \arg(z) \in [\theta(0), \theta(0) + \phi]\},$$

which is entirely in H by the definitions of ϕ and Φ.

The next step is to show that the image of any point with $\nu > \bar{\nu}$ is also within H. Since H is convex and $0 \in H$, it suffices to show that for any ν, $\eta r(\nu) \leq r(\psi)$ where $\theta(\psi) = \theta(\nu) + \phi$. (Note that $\psi > 0$ since $\nu > \bar{\nu}$.) But $\psi < \nu$ as θ is decreasing, whence $r(\nu) \leq r(\psi)$ since r is decreasing. The result follows since $\eta \leq 1$.

The special case (4.51) follows when $\beta = \pi/2$, $\Phi = 0$. □

Lemma 4.4.2. *Define*

$$f(\theta) = \frac{6}{36 + 13\theta^2 + \theta^4}\left(-5\cos(\theta) + (\theta^2 - 6)\frac{\sin(\theta)}{\theta}\right).$$

It holds that

$$\min_{\theta \geq 0} f(\theta, 3) = -11/6. \tag{4.52}$$

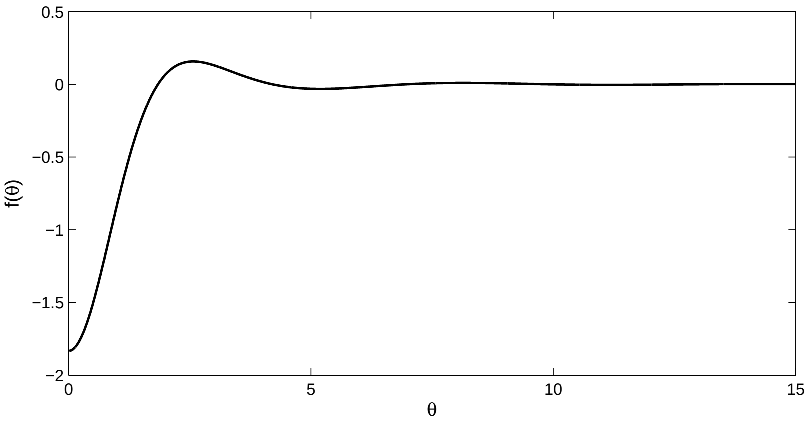


Figure 4.10: Plot of $f(\theta)$.

Proof. First consider when $\theta \geq \sqrt{6}$, for this case

$$\begin{aligned} f(\theta) &\geq \frac{6}{36+13\theta^2+\theta^4}\left(-5-(\theta^2-6)\right) \geq \frac{6(1-\theta^2)}{36+13\theta^2} \\ &= \frac{6(49-(36+13\theta^2))}{13(36+13\theta^2)} \geq -\frac{6(36+13\theta^2)}{13(36+13\theta^2)} = -\frac{6}{13} > -\frac{11}{6}. \end{aligned}$$

Now, when $0 < \theta < \sqrt{6}$,

$$f(\theta) \geq \frac{6}{36+13\theta^2+\theta^4}\left(-5+(\theta^2-6)\right) \geq \frac{6(-11)}{36+13\theta^2+\theta^4} \geq \frac{6(-11)}{36} = -\frac{11}{6}.$$

Finally, study when $\theta \to 0$, knowing that $\lim_{\theta\to 0}\sin(\theta)/\theta = 1$ it is trivial to see that

$$\lim_{\theta\to 0} f(\theta) = -\frac{11}{6}.$$

□

A plot of $f(\theta)$ is shown in Figure 4.10, we observe that the minimum of $f(\theta)$ is achieved at $\theta = 0$, in line with Lemma 4.4.2.

It is now possible to show that the FAST TCP model (4.50) is stable in single-bottleneck networks.

Theorem 4.4.3. *A system with loop gain $L(s)$ given by (4.50) with $\kappa_n = 3$ is stable for arbitrary $\alpha_n > 1$, $0 \leq b < \tau_n$, for all $n = 1, \ldots, N$, and arbitrary $c > 0$, $0 \leq x_c < c$, if for all n*

$$0 < \gamma_n < 6/11. \tag{4.53}$$

Proof. Let

$$G(\theta) = \frac{e^{-j\theta}}{j\theta} \cdot \frac{1+j\theta}{(1+j\theta/3)(1+j\theta/2)}$$

and

$$G_{\tau,\gamma}(\theta) = G(\theta)\frac{\tau - b}{\tau + \gamma b\tau/(j\theta)}$$

so that (4.50) becomes

$$L(j\omega) = \lambda \sum_{n=1}^{N} \frac{\mu_n}{\tau_n} \cdot \frac{\gamma_n\, G_{\tau_n,\gamma_n}(\omega\tau_n)}{j\omega + \lambda\hat{\tau}}.$$

Further, let

$$F(\theta,\kappa) = \frac{2\kappa}{(4+\theta^2)(\kappa^2+\theta^2)}\left(-(2+\kappa)\cos(\theta) + (\theta^2 - 2\kappa)\frac{\sin(\theta)}{\theta}\right).$$

Noting that $\mathrm{Re}(G(\theta)) = F(\theta,3) = f(\theta)$, Lemma 4.4.2 implies

$$G_{\min} \equiv \min_{\theta\geq 0} \mathrm{Re}(G(\theta)) \geq -\frac{11}{6}.$$

By Lemma 4.4.1, $G_{\min} \leq G_{\min,n} \equiv \min_{\theta\geq 0}\mathrm{Re}(G_{\tau_n,\gamma_n}(\theta))$. Letting $F_n(j\omega) = G_{\tau_n,\gamma_n}(\omega\tau_n)$, we now have that the system is stable by Lemma 4.1.2 if

$$1/\gamma_n \geq -G_{\min} \geq -G_{\min,n} = -\min_{\theta\geq 0}\mathrm{Re}(F_n(j\theta)) > 0,$$

and thus the stability condition (4.53). □

Theorem 4.4.3 does not require γ_n or α_n to be equal for all flows. Individual flows may adjust $0 < \gamma_n < 6/11$ and $\alpha_n \geq 1$

The numerical solution of $\min_\theta F(\theta,\kappa)$ as a function of κ is plotted in Figure 4.11, the curve is monotonically increasing. This suggests that the result of Theorem 4.4.3 holds for all $\kappa_n \geq 3$. We also observe that the phase loss that occurs when decreasing κ seems bad for stability, this corresponds to smoothing the queuing delay samples more.

4.4.5 Robustness with respect to network parameters

Let us next examine how the stability result derived in Section 4.4.4 relates to the stability of the nominal model (4.43).

To investigate the closed-loop stability of the system $L_\circ(s)$ (4.43) using the results of Theorem 4.4.3, we will use the robust stability condition briefly introduced in Section 3.1, see (Zhou *et al.*, 1996).

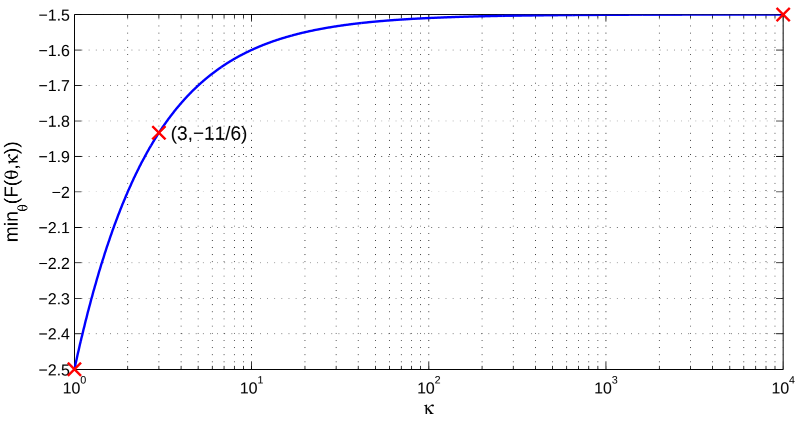


Figure 4.11: Numerical solution of $\min_\theta F(\theta, \kappa)$ as a function of the filter parameter $\kappa \geq 1$.

Robust stability

To be able to apply the robust stability condition $L_\circ$ and L must have the same number of unstable poles. Since for all $h > 0$

$$\lim_{s\to 0} \frac{1-e^{-sh}}{sh} = 1, \tag{4.54}$$

we know that the transfer function of a zero-order hold (4.32) is in $\mathcal{H}_\infty$ and subsequently has no poles in $\bar{\mathbb{C}}^+$. Now Theorem 4.15, and the fact that the poles deriving from the window update and the queuing delay estimation are stable for all feasible parameters, gives that $L_{\circ,n}(s)$ is stable for all $n = 1, \ldots, N$. We can thus conclude that $L_\circ(s)$ is stable. Furthermore, it is trivial to see that $L(s)$ is stable for feasible parameters.

Let us now define the relative model error

$$\Delta_L = \frac{L_\circ - L}{L}, \tag{4.55}$$

and the complementary sensitivity function

$$T = \frac{L}{1+L}.$$

Since $L_\circ$ is open loop stable, and L is open and closed loop stable we have according to the robust stability condition that if

$$|\Delta_L(j\omega)T(j\omega)| = \left| \frac{L_\circ(j\omega) - L(j\omega)}{1 + L(j\omega)} \right| < 1, \quad \text{for all} \quad \omega \geq 0, \tag{4.56}$$

we have that $L_\circ$ is closed loop stable as well under the conditions of Theorem 4.4.3. Note, however, that the converse is not true.

Protocol	Network
$\alpha_1 = 200$	$d_1 = 100\,\text{ms}$
$\alpha_2 = 200$	$d_2 = 110\,\text{ms}$
$\gamma_1 = 0.5$	$c = 60096\,\text{pkts/s}$
$\gamma_2 = 0.5$	$x_c = 0\,\text{pkts/s}$
$\kappa_1 = 3$	
$\kappa_2 = 3$	

Table 4.1: Basic protocol and network parameter configuration for the robustness case study.

Two flow case study

We will consider two window based sources sending over a single bottleneck link and check the condition (4.56) for different network parameters: the capacity c, the amount of cross traffic x_c, and the propagation delay d_1 and d_2.

The protocol parameters for the two flows are set to $\alpha_1 = \alpha_2 = 200$, $\gamma_1 = \gamma_2 = 0.5$ and $\kappa_1 = \kappa_2 = 3$. The basic network configuration is $d_1 = 100$ ms, $d_2 = 110$ ms, 60096 pkts/s (corresponding to a capacity of 500 Mbit/s and a packet size of 1040 byte) and no cross traffic, i.e., $x_c = 0$. The parameter setting is summarized in Table 4.1.

The upper plot in Figure 4.12 shows the magnitude plot of $|\Delta_L(j\omega)T(j\omega)|$ and the lower graph the maximum $\max_\omega |\Delta_L(j\omega)T(j\omega)|$ for different capacities c. Clearly (4.56) is fulfilled. This is the case even that $\gamma_n b/\tau_n \ll 1$ is not fulfilled. For small capacities $c \approx 10000\,\text{pkts/s}$ we have $\gamma_n b/\tau_n \approx 0.485$ in this case, implying that the approximation of the window control time constant (4.46) is coarse. However due to that the robust stability condition is fulfilled we know it is sufficiently good and thus valid.

The graphs in Figure 4.13 are similar plots but for different cross traffic load x_c. Once again the robust stability criterion (4.56) is fulfilled, and once again while we (for large values of x_c/c) are violating the condition we based the window time constant approximation on.

If we instead change the propagation delays while keeping the ratio $d_2/d_1 = 1.1$ we get the plots in Figure 4.14. We observe the same behavior as for the previous two cases.

We have seen this far that the system seems to be robust to changes in parameters c, x_c and changes in d_1 and d_2 with d_2/d_1 a (small) constant. That robustness breaks down when round trip times are heterogeneous is illustrated next.

Let us keep $d_1 = 10$ ms while increasing d_2. Note that the equilibrium queue is $b = 6.7$ ms all the time, and that τ_2/τ_1 grows linearly with d_2. The result is displayed in Figure 4.15. Obviously for this scenario the robust stability condition (4.56) is violated as d_2/d_1 (and thus τ_1/τ_2) increases. Closed loop stability of $L_\circ$ under these circumstances is therefore not guaranteed and it is motivated to study

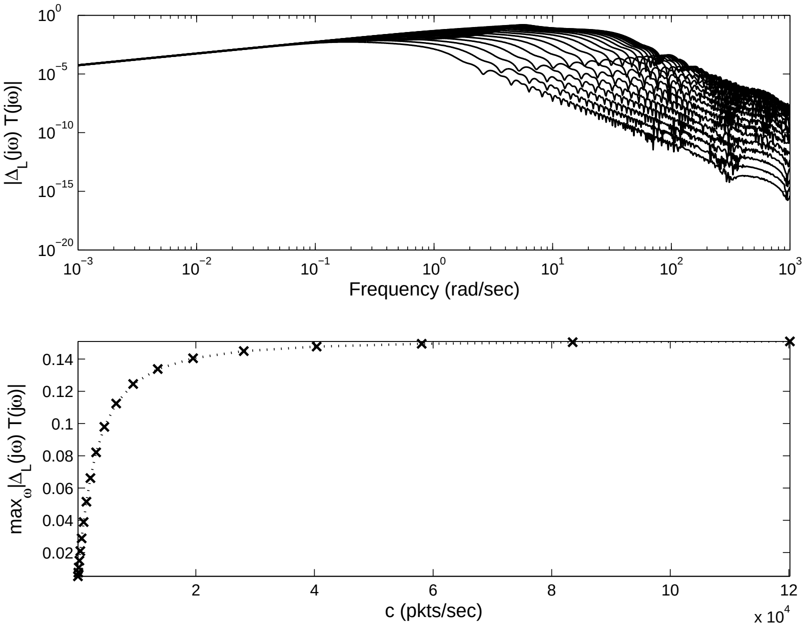


Figure 4.12: Robust stability evaluation. Varying the capacity c.

this case in more detail.

Before proceeding with the analysis let us investigate if we can expect $L_\circ$ to be stable under the conditions of Theorem 4.4.3. We will do that by a numerical example with widely spread RTTs.

Figure 4.16a shows the Nyquist curve of $L_\circ(j\omega)$ for the basic configuration but with $d_1 = 10\,\text{ms}$ and $d_2 = 100\,\text{ms}$. Since $L_\circ(j\omega)$ does not encircle $-1 + j0$ and $L_\circ$ is open loop stable, the Nyquist criterion states that the closed loop system is stable. This is in line with our previous findings since according to Figure 4.15 ($d_2/d_1 = 10$) the robust stability condition is fulfilled for this scenario.

However, increasing the delay of the second flow to $d_2 = 200\,\text{ms}$, and thus according to the plot in Figure 4.15 violating the robust stability condition, gives the Nyquist curve in Figure 4.16b. We observe that it encircles $-1 + j0$. Due to the encirclement and the open loop stability of $L_\circ$ we have by the Nyquist criterion that the closed loop system is unstable.

We now know that the stability properties for $L_\circ$ are worse than for the model L.

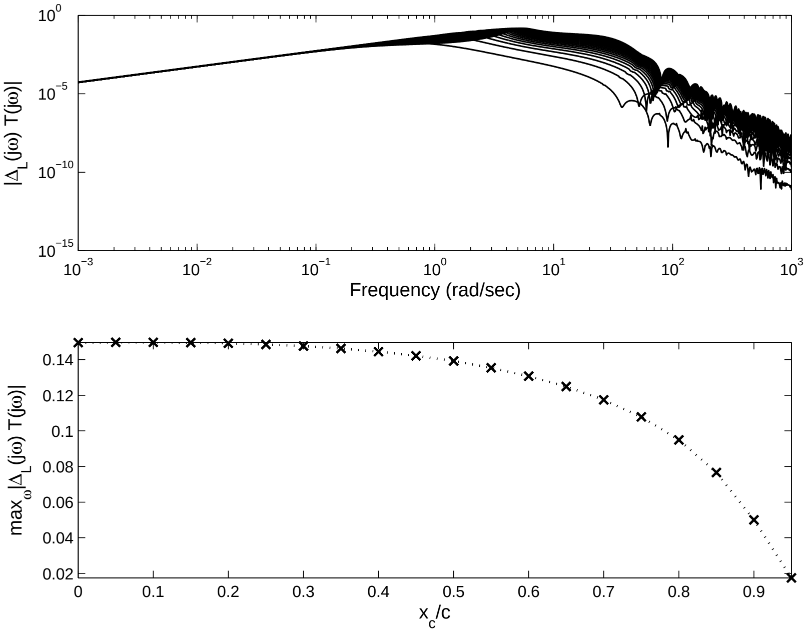


Figure 4.13: Robust stability evaluation. Varying the cross traffic x_c.

4.4.6 Stability revisited

To achieve results on stability we had to reduce model complexity and therefore simplify the original loop gain $L_\circ$ given by (4.43). By a series of approximations we arrived at the the more tractable open loop transfer function L, given by (4.50), which we based the analysis on. However, when investigating the robustness with respect to modeling errors of our results in the previous section, we got indications that this approximation was not sufficiently good in the case of heterogeneously distributed round trip times. At least we could not confirm stability for the full model $L_\circ$ in the scope of Theorem 4.4.3. In particular, we found a numerical example with parameters within the stability limits of Theorem 4.4.3 where the original model $L_\circ$ was not closed loop stable. We will now investigate this in more detail.

Two flows single bottleneck link model under heterogeneous RTTs

Even though we have just seen that the model previously used for stability analysis is not accurate enough for some parameter settings, we would like to avoid working directly with the full loop gain $L_\circ$ due to model complexity. We will hence strive towards an alternative model simplification better tailored for our needs.

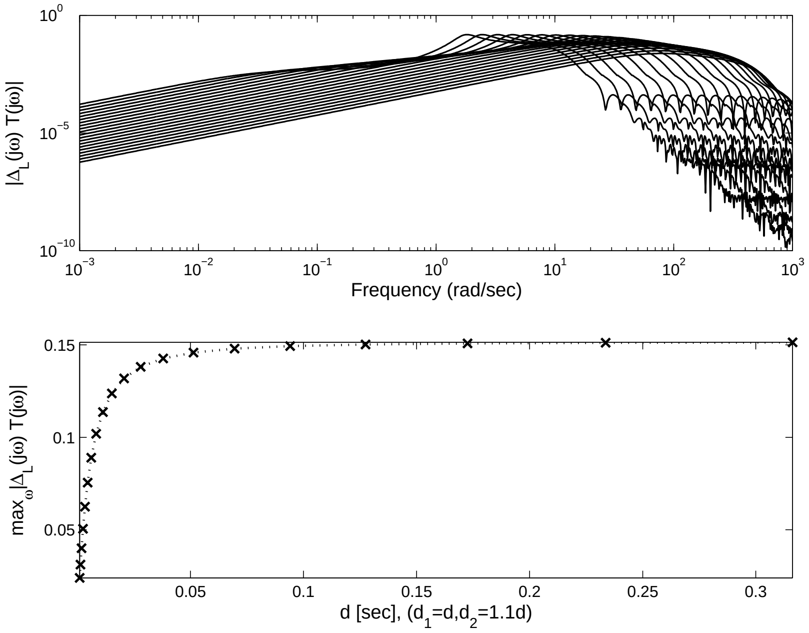


Figure 4.14: Robust stability evaluation. Varying the propagation delay, keeping the ratio $d_2/d_1 = 1.1$ constant.

Our focus is the large RTT heterogeneity case, let us thus for analytical purposes only consider the case $b \ll d_1 \ll d_2$. For this setting, we have that the time constant of the window control, which is about ${\tau_n}^2/(\gamma_n b)$ by (4.33), is substantially larger than the time constant of the estimator dynamics, approximately equal to $\tau_n/\kappa_n = \tau_n/3$ from (4.36). Thus, it is valid to neglect the estimator dynamics in the analysis. Similarly, the approximative time constants of the zero-order holds deriving from the sampling in the window mechanism and the flight size, which is $\tau_n/2$, are much smaller compared to the window control time constant. It is therefore motivated to ignore these as well. It was already in the derivation of L stated that the zero-order hold deriving from the sampling in the estimator can be ignored when the queuing delay b is small.

Study the linear window mechanism (4.33) in a little bit more detail. When $b \ll d_n$, which implies $\tau_n \approx d_n$, we have

$$G^{\circ}_{w_n\hat{q}_n}(s) \approx -\frac{\alpha_n d_n/b^2}{s{\tau_n}^2/(\gamma_n b)+1} \approx -\frac{x_n\tau_n}{s{\tau_n}^2/\gamma_n + b} \approx \frac{\gamma_n x_n}{s\tau_n}. \tag{4.57}$$

The last approximation is good for frequencies greater than about a decade larger than the pole, i.e., $\hat{\omega} > 10\gamma_n b/{\tau_n}^2$. For $b \ll \tau_n$ the cut-off frequencies of the expressions in (4.57) are approximately $\hat{\omega}_c \approx \gamma_n x_n/\tau_n$. Since $\hat{\omega}_c \gg \hat{\omega}$ under our

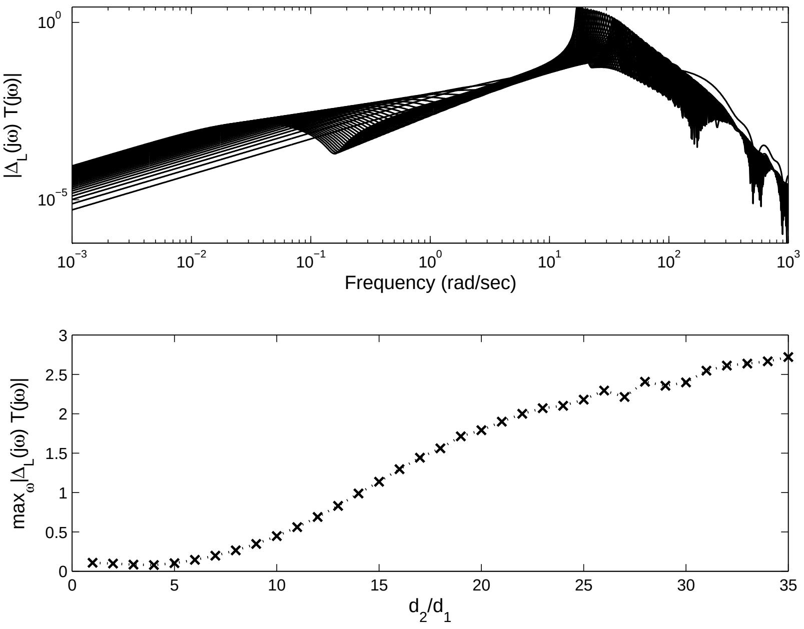


Figure 4.15: Robust stability evaluation. Varying the relative delay d_2/d_1.

small queue assumption this should be sufficiently accurate. The approximations in (4.57) becomes exact in the limit $b \to 0$, c.f., Remark 4.4.1.

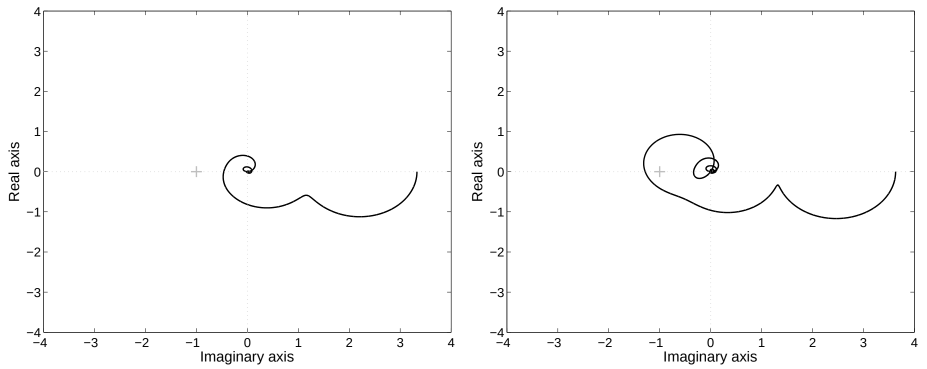
For two flows with equal parameters α we have have equal equilibrium rates $x_1 = x_2 = c/2$ when no cross traffic is present. Furthermore assume for simplicity equal γ in the FAST window control, i.e., $\gamma_n = \gamma$ in (4.57). Ignoring estimator and zero-order holds and using the window control approximation (4.57) and the ACK-clocking (4.40) now yields that the loop gain reduces to

$$\hat{L}(s) = \gamma \frac{\dfrac{e^{-s\tau_1}}{s\tau_1}(1 - e^{-s\tau_2}) + \dfrac{e^{-s\tau_2}}{s\tau_2}(1 - e^{-s\tau_1})}{2 - e^{-s\tau_2} - e^{-s\tau_1}}. \tag{4.58}$$

By recalling Theorem 4.2.2, that states that the ACK-clocking is stable, we know that $\hat{L}$ does not contain any RHP poles outside the origin. Next we will demonstrate the accuracy of this approximation.

The Bode plot in Figure 4.17 shows $L_\circ$ (solid line), L (dotted line) and $\hat{L}$ (dashed line). The parameter configuration is given in Table 4.1 but with $d_1 = 10\,\text{ms}$, $d_2 = 350\,\text{ms}$ and $\alpha_1 = \alpha_2 = 2$. For this large RTT heterogeneity and small queue case we see, as expected, that $\hat{L}$ clearly is a better approximation to $L_\circ$ compared to L, at frequencies relevant for closed loop stability.

If we increase the equilibrium queue size b a factor 100 (by multiplying the α



(a) Delay $d_1 = 10\,\text{ms}$ and $d_2 = 100\,\text{ms}$. (b) Delay $d_1 = 10\,\text{ms}$ and $d_2 = 200\,\text{ms}$.

Figure 4.16: Nyquist curve of $L_\circ$ with parameters according to Table 4.1 except for delays d_1 and d_2.

parameters by 100) we get the Bode plots in Figure 4.18. We observe from the plots that even if the small queue assumption $d_1 \gg b$ is not fulfilled, still $\hat{L}$ seems superior to L around the cross over frequency.

Instability due to RTT heterogeneity

Since each individual flow does not have complete knowledge of the network, we would like to be able to set FAST's parameters, such as γ, so that it will be stable in *all* networks.

It is, however, possible to use the model (4.58) to show that that is impossible. For a given γ, a network carrying two flows with very different RTTs can be constructed such that FAST is unstable. This is concluded from the following theorem.

Theorem 4.4.4. *Consider the transfer function (4.58),*

$$\hat{L}(s) = \gamma \frac{\dfrac{e^{-s\tau_1}}{s\tau_1}(1 - e^{-s\tau_2}) + \dfrac{e^{-s\tau_2}}{s\tau_2}(1 - e^{-s\tau_1})}{2 - e^{-s\tau_2} - e^{-s\tau_1}}.$$

For all $\gamma > 0$ and $\tau_1 > 0$ there exists a $\tilde{\tau} > 0$ such that for all $\tau_2 > \tilde{\tau}$, $\hat{L}(s)$ is unstable.

Proof. See Appendix B. □

Physically, the cause of instability is the feedback on the time scale $\tau_2 \gg \tau_1$ in the ACK-clocking mechanism. In the model (4.58) this appears as arbitrarily large

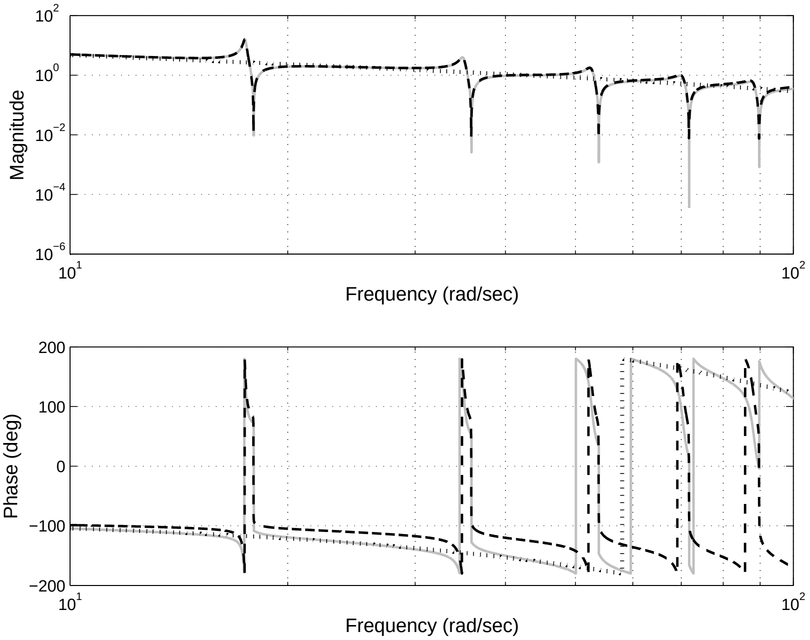


Figure 4.17: Bode plot. Propagation delays $d_1 = 10\,\text{ms}$, $d_2 = 350\,\text{ms}$. Queuing delay $b = 0.067\,\text{ms}$. Solid line: $L_\circ(j\omega)$. Dotted line: $L(j\omega)$. Dashed line: $\hat{L}(j\omega)$.

positive gain near $s = j2\pi/(\tau_1 + \tau_2)$, (cf., (B.4) in Appendix B). For stability both sources should scale down their feedback gain in proportion to the feedback delay. From (4.57) we get that the gain of the nth flow under the given assumptions is approximately

$$\frac{\gamma x_n}{\omega \tau_n},$$

where ω denotes the frequency in radians per second. Flow 1 thus scales down its gain in proportion to its own RTT τ_1. However, it does not compensate for the feedback induced by Flow 2 explicitly. It is unclear how such a down scaling should be done in a decentralized fashion without punishing high RTT flows or explicitly communicating delay information between sources.

The last step in our stability analysis of FAST TCP is to check if our results based on mathematical models matches real experiments. The following numerical results support Theorem 4.4.4. We start with an example using NS-2.

Example 4.4.1 (Instability due to heterogeneous RTTs: NS-2). *Consider two FAST flows with 1040 byte packets sharing a 200 Mbit/s bottleneck, with* $d_1 = 10$ *ms and* $d_2 = 303$ *ms and FAST parameters* $\gamma = 0.5$ *and* $\alpha = 100$ *(Wei et al., 2006). The Nyquist plot of (4.58) in Figure 4.19 encircles* -1*, indicating instability. NS-2*

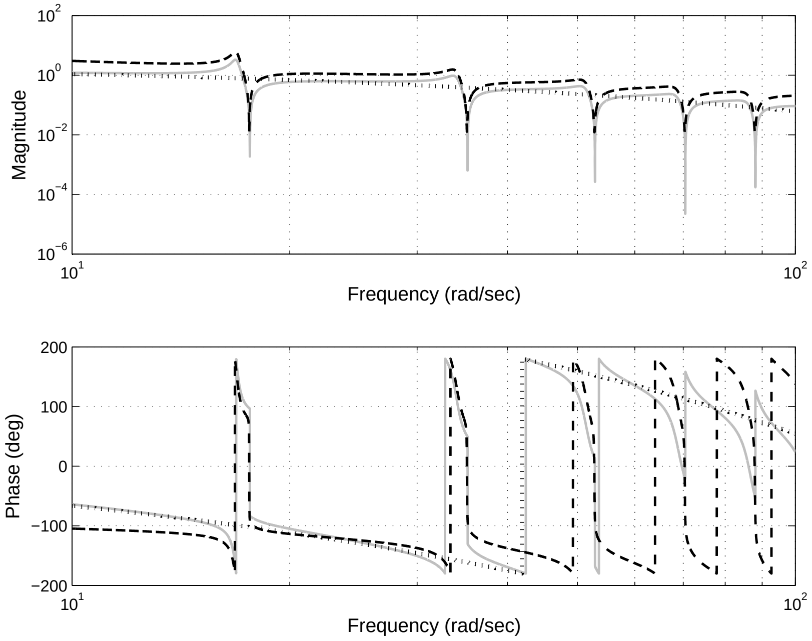


Figure 4.18: Bode plot. Propagation delays $d_1 = 10\,\text{ms}$, $d_2 = 350\,\text{ms}$. Queuing delay $b = 6.7\,\text{ms}$. Solid line: $L_\circ(j\omega)$. Dotted line: $L(j\omega)$. Dashed line: $\hat{L}(j\omega)$.

simulations, reported in the three remaining plots, show that there is indeed sustained oscillation at around $1/d_2 \approx 3\,Hz$*. The variation in* window *size shows that this is not simply packet level sub-RTT burstiness. For this example FAST's multiplicative increase mode was disabled.*

Obviously the NS-2 version of FAST TCP matches our predictions. We will now continue with a real testbed experiment.

Example 4.4.2 (Instability due to heterogeneous RTTs: WAN-in-Lab)**.** *Similarly, consider two FAST flows with 1500 byte packets sharing a 1 Gbit/s bottleneck, with* $d_1 = 6$ *ms and* $d_2 = 130$ *ms and FAST parameters* $\gamma = 0.5$ *and* $\alpha = 30$ *packets. The Nyquist plot of (4.58) for this configuration is shown in Figure 4.20. It encircles* -1 *and thus we expect the system to be unstable. The system was implemented in WAN-in-Lab (Lee et al., 2007a). Figure 4.21 shows that there is again sustained oscillation of over 50 packets at around* $1/d_2 \approx 8\,Hz$*, indicating instability. Figure 4.21 shows that there is again sustained oscillation at around* $1/d_2 \approx 8\,Hz$*. Since it is difficult to measure the queue size inside a hardware router, the plots instead show oscillation in the window size of the flow with a 6 ms RTT. The sustained oscillations of over 50 packets illustrates the instability. The windows were sampled at 20 samples/second.*

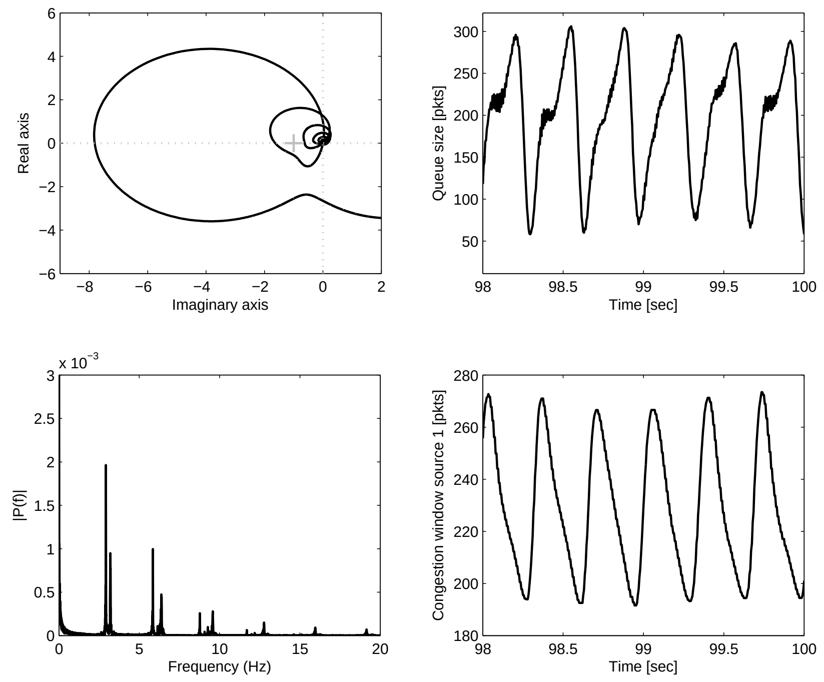


Figure 4.19: Instability of FAST due to heterogenous RTTs: $d_1 = 10\,\mathrm{ms}$, $d_2 = 303\,\mathrm{ms}$. Top left: Nyquist plot of loop gain (4.58). Top right: Bottleneck queue size. Lower left: Magnitude spectrum (FFT) of queue size, without DC component. Lower right: Window size, source 1.

The reason for the two brief metastable periods near 500 and 1000 seconds is not clear. However, the increase in throughput during those times shows the harmful effects of even bounded instability, and importance of investigating the stability of protocols.

The lower two subfigures show closeups of the abatement and onset of stability. Similar experiments with two flows of 6 ms do not exhibit this oscillation, indicating that it is indeed due to the heterogeneity.

FAST's multiplicative increase mode was used only when the measured delay was less than 0.3 ms.

4.5 Summary

In this chapter we have shown that the ACK-clocking mechanism is linearly stable around its unique equilibrium. However, in the case of rational ratios between flows' RTTs the system may not be internally stable in the sense that sources' rates are

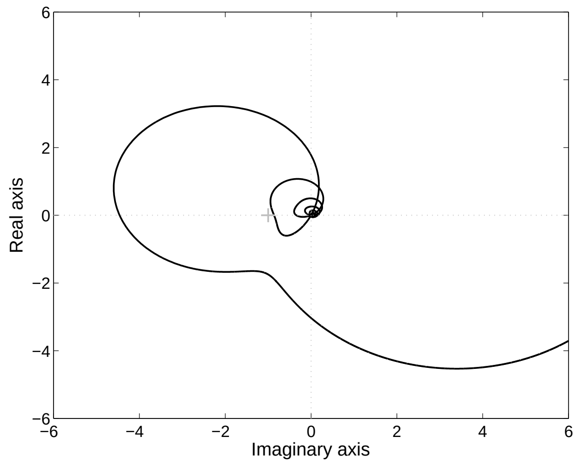


Figure 4.20: Nyquist curve of (4.58): $d_1 = 6\,\mathrm{ms}$, $d_2 = 130\,\mathrm{ms}$, $c = 1\,\mathrm{Gbit/s}$, $x_c = 0$, $\gamma = 0.5$ and $\alpha = 30$ packets.

oscillating while the queue remains constant. Furthermore, we discussed different strategies how to reduce the complexity of the ACK-clocking model.

The quantitative difference between protocol window and flight size was also investigated. We found that for the typical case when the window size is updated once per RTT, the difference was in the order of an RTT.

Finally, a linear model of FAST TCP sending over a single bottleneck link was derived, simplified and used for analysis. Using this simplified model, valid for moderately hetergoeneously distributed RTTs, we showed that the system is stable for default parameter values. We observed, however, that the model accuracy degenerated as the RTT heterogeneity increased. Thus, using an alternative model accurate for such scenarios we proved that for any fixed protocol parameters the system will destabilize for some distributions of RTTs. The theoretical results was also confirmed with NS-2 simulations and testbed experiments.

4.6 Related work

Window based congestion control, more specifically TCP, is ubiqitous on the Internet today and has been so since introduced in the late 1980s. Throughout the years a massive amount of research has been spent on trying to understand the static and dynamical properties of the protocol, see, e.g., (Mathis *et al.*, 1997; Lakshman and Madhow, 1997; Padhye *et al.*, 1998; Low, 2000; Johari and Tan, 2001). Numerous new designs and improvements has been proposed, see, e.g., (Mathis *et al.*, 1996; Ludwig and Katz, 2000; Gerla *et al.*, 2001) and the discussion in Section 2.2.4. In relation to the huge research effort devoted to TCP (as of April 1, 2008 an ISI

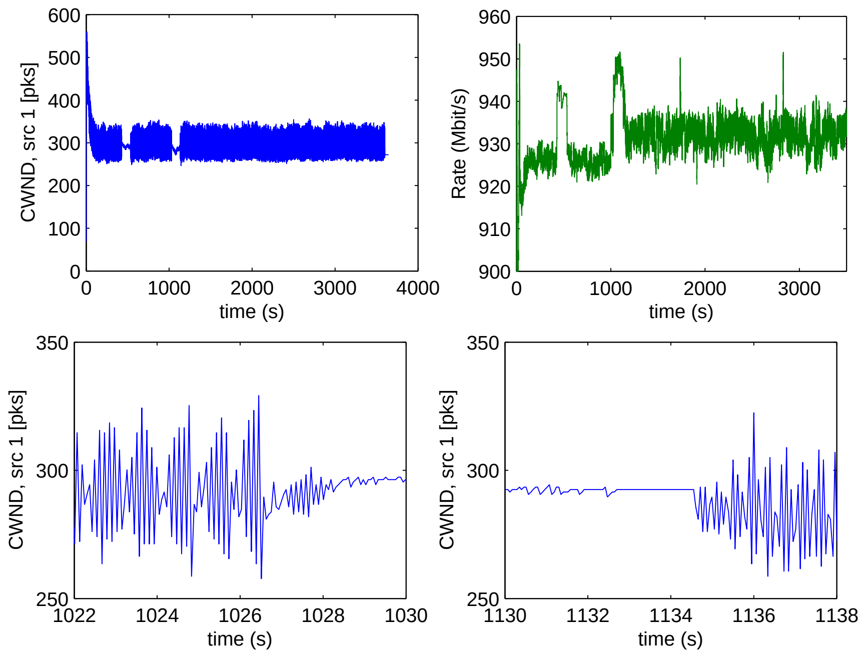


Figure 4.21: Instability of FAST due to RTT heterogeneity: $d_1 = 6\,\text{ms}$, $d_2 = 130\,\text{ms}$. Upper left: Window size, source 1, w_1. Upper right: Total received rate. Lower left: Oscillation abatement, w_1. Lower right: Oscillation onset, w_1.

search on "TCP congestion control" gives 839 hits) it is rather suprising how little work that exists on the dynamics of the ACK-clocking mechanism, the fundament of all window based congestion control.

In his seminal paper, Jacobson (1988) states that the ACK-clocking should be stable in the sense that quantities remain bounded: "By 'conservation of packets' we mean that for a connection 'in equilibrium', i.e., running stably with a full window of data in transit, the packet flow is what a physicist would call 'conservative': A new packet isn't put into the network until an old packet leaves. The physics of flow predicts that systems with this property should be robust in the face of congestion." Then, with the recent congestion collapses in mind, he makes the observation that the Internet was not robust at all and turn the focus to the design of the window control algorithm. The focus of the networking community seems to have remain there since. However, the resource allocation properties of the ACK-clocking mechanism has been studied in (Massoulié and Roberts, 2002), the same results also appears in flight in (Wei *et al.*, 2006). Stability of the ACK-clocking mechanism has been studied, using cruder models than used in this work, in (Möller *et al.*, 2006; Jacobsson *et al.*, 2006; Möller, 2008). Results are, however, qualitatively the same as here.

In (Wei, 2007) it is pointed out that the Internet research community has overlooked the microscopic behavior of TCP for years and that the ACK-clocking mechanism actually has a significant impact on the macroscopic performance of both loss and delay based TCPs. This is also the topic of the paper (Tang *et al.*, 2008), which highlights the need for incorporating important microscopic timescale effects in macroscopic fluid flow models. The bursty pattern of acknowledgments caused by bursty transmission will cause the transmissions in the next RTT also to be bursty. This self-perpetuating burstiness is due to the ACK-clocking and has been verified with simulations and real Internet tests (Jiang and Dovrolis, 2005).

The techniques used to analyze the stability of FAST TCP in Section 4.4 are significantly different from the techniques in the existing literature on linear stability of TCP, in two respects. First, the usual approach is to deal with the loop gain $L(s)$, that appears as a weighted sum of transfer functions $L_n(s)$, is to find a convex hull that contains all individual $L_n(j\omega)$ curves and then argue that any convex combination of them is still contained by the convex hull. See, e.g., (Vinnicombe, 2002; Low *et al.*, 2003; Choe and Low, 2003). However, the proof of Lemma 4.1.2 deals directly with $L(j\omega)$ instead of $L_n(j\omega)$. Second, for each ω, a separate region is found to bound $L(j\omega)$ away from the interval $(-\infty, -1]$. That is, the half plane $H(\omega)$ defined by (4.9) depends on ω. In existing works, convex regions are typically used to bound the whole curves and hence are independent of ω. One exception is (Paganini *et al.*, 2005), where the frequency range is divided into two parts and different convex regions are used in the two parts. These two features lead to tighter bounds, which is necessary for the analysis of the problem in Section 4.4. The method here is somewhat related to the work appearing in (Jönsson *et al.*, 2007), which also explores the idea of using frequency dependent separating hyperplanes to assess stability.

Stability of FAST TCP has been studied several times. Using a static link model, only accounting for the static gain in the ACK-clocking mechanism, it has been shown theoretically that FAST TCP flows are always stable in the presence of delays for the case of homogeneous sources and feasible parameters (Choi *et al.*, 2005; Wang *et al.*, 2004, 2005). Using a refined link model that captures the RTT hetereogeneity to a larger extent than the static link model it is shown that the stable parametric region is smaller than specified in (Wei *et al.*, 2006). The examples in Section 4.4.4 are the first demonstrating instability of FAST TCP, they appeared in (Tang *et al.*, 2008). That previous experimental work failed to find unstable configurations is likely due to that it did not explore cases with sufficient heterogeneity in feedback delays. In addition to the instability examples discussed above, the paper (Tang *et al.*, 2008) also analyzes and demonstrates in simulations instability modes of FAST TCP due to RTT synchronization.

For stability analysis of other congestion control schemes we refer to the discussion and the references in Section 2.5.4.

For an introduction to linear systems theory, see (Rugh, 1996). Frequency domain analysis design methods can be found in (Zhou *et al.*, 1996). Convex analysis and optimization is discussed thouroughly in (Boyd and Vandenberghe, 2004) and (Bertsekas *et al.*, 2003).

Chapter 5

Experimental results and validation

IN engineering and science mathematical models are widely used as a vehicle to study physical systems. For reliable predictions, however, it is crucial to establish the validity of the models. Proper model validation is therefore of great importance in this type of work. In this chapter the models derived in Chapter 3 and Chapter 4 are validated.

5.1 Experiment design

Validation experiments may be executed in open as well as closed loop. We will here do both depending on what is most convenient.

5.1.1 Open loop

Validation experiments in open loop are in principal executed as follows. Consider a system $G : u \to y$ and a (possibly nonlinear) model of the system $\hat{G} : \hat{u} \to \hat{y}$, where system and model inputs are of the same dimension, the same holds for outputs.

Inputs u will naively be chosen as impulses, steps or sinusoids; optimal choices of inputs are not in the scope of this thesis.

An input u will be applied to the system G, producing an output y. The same input u will be fed into the model $\hat{G}$, resulting in an output $\hat{y}$. The predicted output $\hat{y}$ is then simply compared to the true output y by plotting them in the same diagram.

Example 5.1.1. *Consider a system*

$$G(s) = \frac{1}{s+1},$$

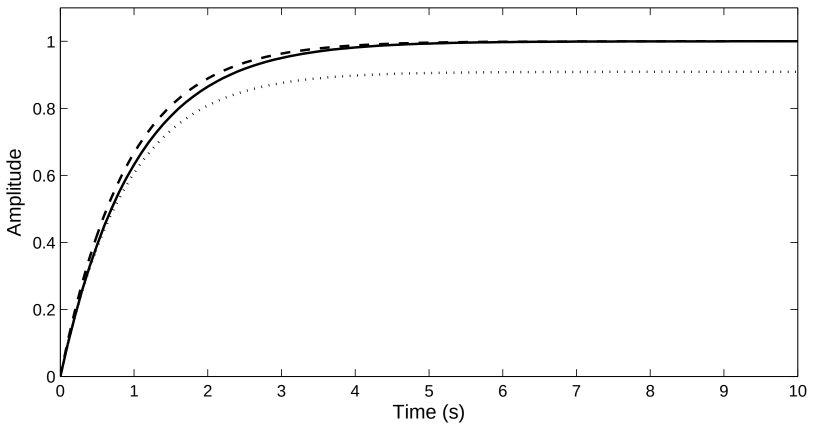


Figure 5.1: Open loop validation example. Step responses. Solid line: True system G. Dashed line: Model $\hat{G}_1$. Dotted line: Model $\hat{G}_2$.

modeled with

$$\hat{G}_1(s) = \frac{1}{s/0.9 + 1},$$

and

$$\hat{G}_2(s) = \frac{1}{s + 0.9}.$$

The true system G is subject to a step in the input signal. The output is the solid line in Figure 5.1. The input signal is collected and used as input signals in simulations with $\hat{G}_1$ and $\hat{G}_2$. The outputs of the two simulations corresponds to the dashed line and dotted line in Figure 5.1 respectively. We observe that $\hat{G}_1$ shows better agreement with G compared to $\hat{G}_2$, subsequently the naive conclusion is that it seems superior.

We remark that a model that may the better choice than another for open loop simulations, may actually be inferior in, e.g., a control design, cf., Example 3.1.1. We will here, however, not quantify model accuracy to a higher extent than visual comparison with the true system.

5.1.2 Closed loop

It is well known that in closed loop identification, for certain types of excitation and certain methods there is a risk of identifying the inverse of the regulator instead of the desired dynamics, or a linear combination of this dynamics and the desired

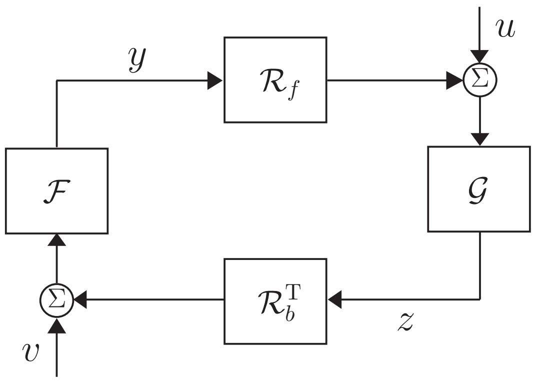


Figure 5.2: Closed loop system with linear dynamics $\mathcal{F}$ and G, and external excitation signals u and v.

dynamics. When producing validation data in closed loop, excitation must thus be chosen with care to avoid misleading results. We will illustrate this for the closed loop system in Figure 5.2 with corresponding linear feedback configuration

$$y = \mathcal{F}\left(\mathcal{R}_b^{\mathrm{T}} z + v\right), \tag{5.1a}$$

$$z = \mathcal{G}\left(\mathcal{R}_f y + u\right), \tag{5.1b}$$

when it is excited with excitation signals u and v.

System excitation

Suppose that we are able to collect experimental data y and z. When the system is excited through u only, we have in the Laplace domain

$$y = \left(I - \mathcal{F}\mathcal{R}_b^{\mathrm{T}}\mathcal{G}\mathcal{R}_f\right)^{-1}\mathcal{F}\mathcal{R}_b^{\mathrm{T}}\mathcal{G}u, \tag{5.2}$$

$$z = \left(I - \mathcal{G}\mathcal{R}_f\mathcal{F}\mathcal{R}_b^{\mathrm{T}}\right)^{-1}\mathcal{G}u. \tag{5.3}$$

By use of the matrix identity $(I + AB)^{-1}A = A(I + BA)^{-1}$ it is evident from (5.2) and (5.3) that

$$y = \mathcal{F}\mathcal{R}_b^{\mathrm{T}}\left(I - \mathcal{G}\mathcal{R}_f\mathcal{F}\mathcal{R}_b^{\mathrm{T}}\right)^{-1}\mathcal{G}u = \mathcal{F}\mathcal{R}_b^{\mathrm{T}} z.$$

Clearly, the closed loop data $\{y, z\}$ from an experiment with excitation only in u describes the dynamics $\mathcal{F}\mathcal{R}_b^{\mathrm{T}}$, thus any attempt identifying $\mathcal{R}_f\mathcal{G}$ will fail in this case.

Similarly, by exciting the system by means of the external signal v instead we get

$$\begin{aligned} y &= \left(I - \mathcal{F}\mathcal{R}_b^{\mathrm{T}}\mathcal{G}\mathcal{R}_f\right)^{-1}\mathcal{F}v, \\ z &= \left(I - \mathcal{G}\mathcal{R}_f\mathcal{F}\mathcal{R}_b^{\mathrm{T}}\right)^{-1}\mathcal{G}\mathcal{R}_f\mathcal{F}v; \end{aligned}$$

and furthermore, like for the previous case

$$z = \mathcal{G}\mathcal{R}_f\left(I - \mathcal{F}\mathcal{R}_b^{\mathrm{T}}\mathcal{G}\mathcal{R}_f\right)^{-1}\mathcal{F}v = \mathcal{G}\mathcal{R}_f y.$$

Naturally, we observe that $\mathcal{R}_f\mathcal{G}$ is possible to identify in this experiment configuration, in contrast to $\mathcal{F}\mathcal{R}_b^{\mathrm{T}}$, which is not. This is now illustrated with an example.

Example 5.1.2. *Consider a closed loop system according to Figure 5.2 with open loop transfer functions*

$$\begin{aligned} \mathcal{F}(s) &= \frac{1}{s+1}, \\ \mathcal{G}(s) &= \frac{1}{s+3}, \end{aligned}$$

and, for simplicity, let $\mathcal{R}_f = \mathcal{R}_b = 1$.

First, let v be subject to a unit step at time 1 s. The solid lines in the upper plots of Figure 5.3 correspond to data from a simulation of the system, the left plot shows y and the right plot z. The dashed line is the simulated output signal, more specifically $\hat{y}$ is the output of an open loop simulation of $\mathcal{F}$ with z as input signal, i.e., $\hat{y} = \mathcal{F}z$. The dotted line is the simulated output signal $\hat{z} = \mathcal{G}y$. We observe that $\hat{z}$ agrees with z and thus the data set $\{y, z\}$ represents the dynamics given by $\mathcal{G}$ as predicted.

Analogously, let u be subject to a unit step at time 1 s. The solid lines in the lower plots of Figure 5.3 correspond to data from a simulation of this scenario, the left plot displays y and the right plot z as before. For this case, obviously $\hat{y}$ and y coincides whence the data $\{y, z\}$ corresponds to the dynamics $\mathcal{F}$.

There exist a wide variety of identification methods (Ljung, 1999). However, to maintain consistent estimates when the system is operating in closed loop it is essential to chose identification method with care. A well-known method is the Prediction Error method that estimates model parameters which minimizes the one-step-ahead output prediction error. To avoid bias when the Prediction Error Method is used it is necessary that the true dynamics is within the chosen model structure and also that the true noise dynamics is included in the given noise description. When the external input also is known, an indirect method such as the Two-Stage Method (Van den Hof and Schrama, 1995) allows consistent estimation even without a noise model. All these methods work when there is excitation both in u and v.

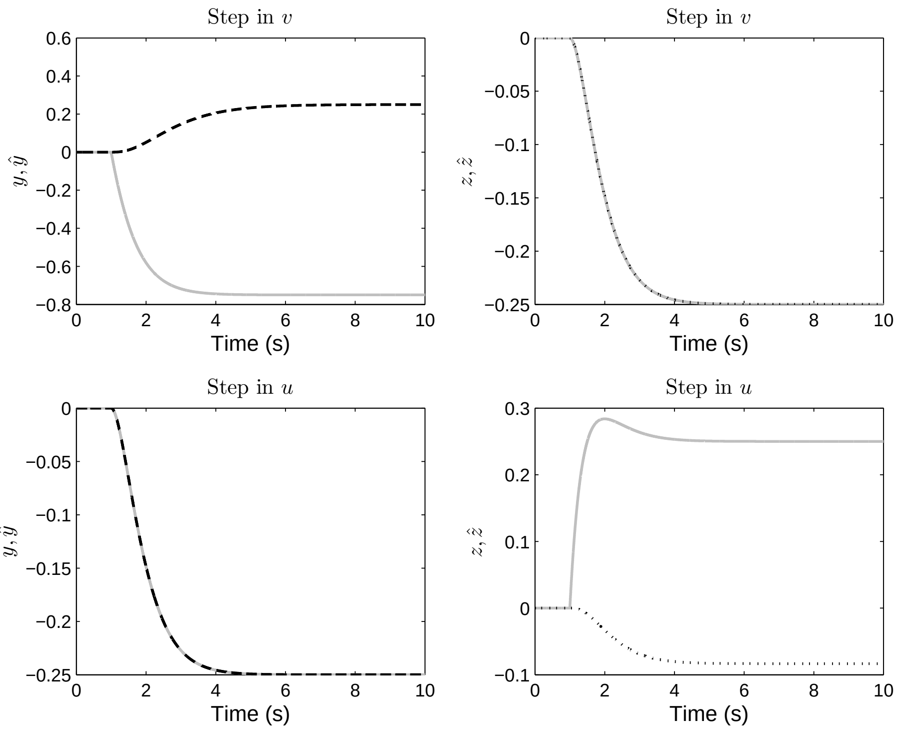


Figure 5.3: Closed loop identification example. Solid lines: "True" closed loop data y and z. Dashed lines: Model data $\hat{y} = \mathcal{F}z$. Dotted lines: Model data $\hat{z} = \mathcal{G}z$.

Excitation in practice

Let us relate what we know about the closed loop configuration given by (5.1) and Figure 5.2 to a system of window based sources utilizing a network, cf., Figure 5.4.

Let y represent the flight size, z the link price (queuing delay), $\mathcal{R}_f$ and $\mathcal{R}_b$ forward and backward propagation delay, $\mathcal{F}$ the protocol dynamics (flight, window, estimator), and finally $\mathcal{G}$ the link dynamics. In this context it is realized from the discussion in the previous section that for a scenario where sources' flight sizes f (y) and link prices p (z) are collected and the network delay configuration is known, it is possible to

- identify sources' congestion control protocol dynamics by means of external changes in the input to the links;
- identify the link dynamics by disturbing the price signal seen by the sources.

So, if generation of validation data is the objective, how should external excitations be realized in practice in a network bed or using packet level simulations with, e.g., NS-2 or similar? As just stated, in the identification of the source dynamics

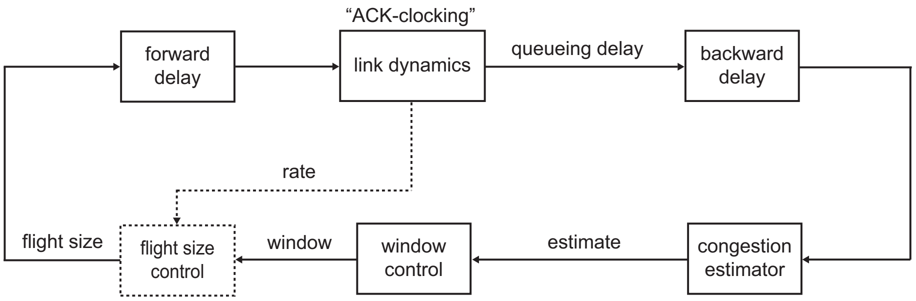


Figure 5.4: Block diagram of the control loop from the perspective of an individual window based congestion control protocol.

the link input should be manipulated externally. This is easily done by sending suitable cross traffic over the links, e.g., by means of UDP protocols or similar. Note that when considering disturbances from an equilibrium point, the system should be excited through changes in the cross traffic.

If we are seeking the link dynamics and thus would like to perturb the prices fed back to the sources, applying external cross traffic is not suitable. It is more convenient to directly manipulate the internal price variable in the source algorithm. However, when it comes to validation experiments of the link dynamics it turns out it is simple to break the loop and produce data in open loop. Subsequently, this will be preferred.

5.2 ACK-clocking

In this part we will validate the ACK-clocking model (3.10) that was derived in Section 3.3. The model is simulated in Simulink, and the simulation output is compared with packet level data achieved using the NS-2 simulator. Some validation examples using a physical testbed is also provided here. As well as validating the model, the results comparing all three systems demonstrate that NS-2 is sufficiently accurate to use as a reference in this type of work. Details about the testbed and additional testbed experiments are given in Appendix C.

Recapture the window based congestion loop model introduced in Chapter 3 and displayed in Figure 5.4. All validation experiments will be performed in open loop, i.e., breaking the loop in Figure 5.4 by disabling the window control and only executing positive changes of the flight size $f(t)$, the latter to decouple the ACK-clocking from the flight size dynamics. Recall that it is impossible for a congestion control algorithm to decrease the number of packets in flight faster than the rate at which ACKs arrive, so step decreases in $f(t)$ are not meaningful.

First, the ACK-clocking model (3.10) will be compared quantitatively with NS-2 for a single bottleneck link. Subsequent multiple bottleneck experiments follows.

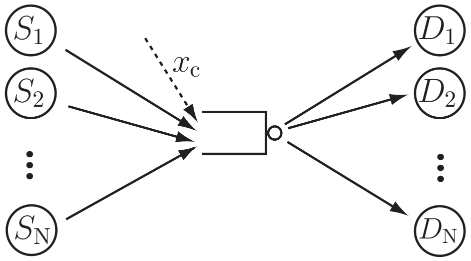


Figure 5.5: Single link configuration. The ith source destination pair is represented by S_i and D_i.

The testbed examples concludes this part.

5.2.1 Single link network

The topology for the single link bottleneck experiments is given in Figure 5.5, S_i represents the location of the ith source and D_i the destination where the receiver reside. In the figure the source/destination pairs may be window based as well as non-window based (and thus corresponds to “cross traffic”). We will for the single link case focus on the case of two window based sources.

Identical RTTs

A scenario where a system of two window based flows sending over a single bottleneck link with capacity 150 Mbit/s is considered. Packet size is $\rho = 1040$ byte. Propagation delays are $d_1 = d_2 = 100\,\mathrm{ms}$, and flight sizes are initially set to $f_1 = f_2 = 990$ packets. After convergence, at 25 seconds, f_1 is increased stepwise to 1090 packets.

The solid line in Figure 5.6 shows the bottleneck queue size (in seconds) when this scenario is simulated in NS-2, we observe an immediate response to the flight size step in the queue. The dashed line is the DAE model (3.8) which captures this behavior.

Identical RTTs and cross traffic

For an identical scenario as described above but with capacity $c = 750\,\mathrm{Mbit/s}$ and constant cross traffic $x_c = 600\,\mathrm{Mbit/s}$ we get a queue response according to Figure 5.7. The solid line is the NS-2 simulation output and the dashed line the DAE model (3.8). The DAE model is very accurate and even captures the burstiness in the system leading to the step-wise changes in the queue. When cross traffic is present the transient is significant.

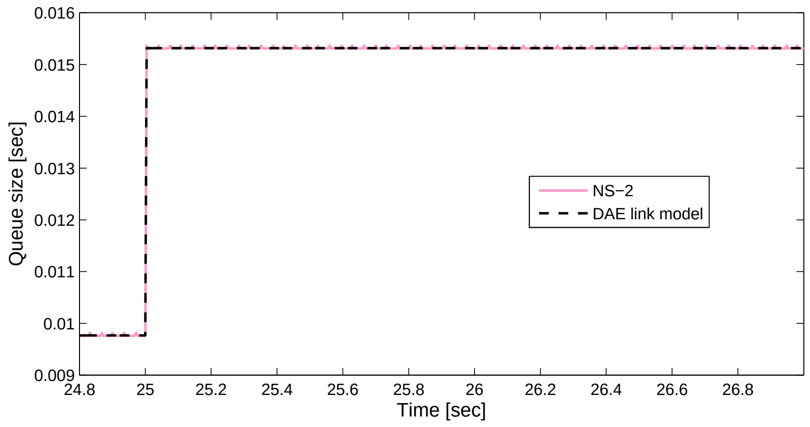


Figure 5.6: Single bottleneck. Two flows with $d_1 = d_2 = 100\,\text{ms}$. No cross traffic. Flight size of short RTT flow increases.

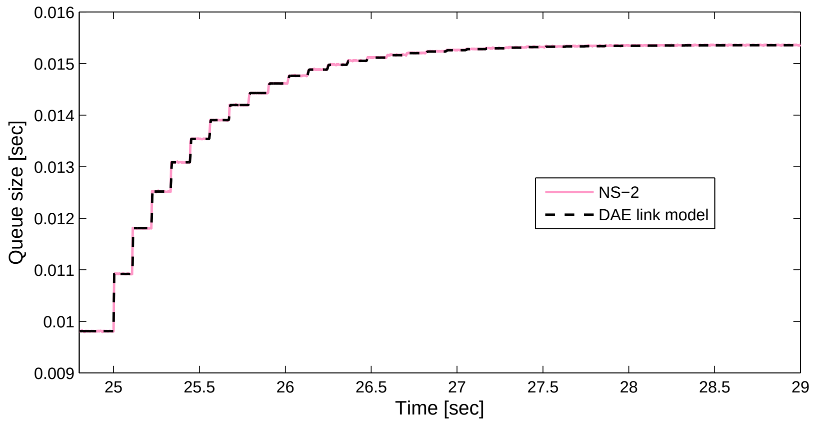


Figure 5.7: Single bottleneck. Two flows with $d_1 = d_2 = 100\,\text{ms}$. Cross traffic of 80% of link capacity present. Flight size of short RTT flow increases.

Heterogeneous RTTs

Consider a system of two window based flows sending over a single bottleneck link with capacity 150 Mbit/s, with $\rho = 1040$ byte packets, where the sources' flight sizes are kept constant. Propagation delays are $d_1 = 10\,\text{ms}$ and $d_2 = 190\,\text{ms}$. The flight sizes are initially $f_1 = 210$ and $f_2 = 1500$ packets respectively. After convergence, at 25 seconds, f_1 is increased step-wise from 210 to 300 packets.

The solid line in Figure 5.8 shows the bottleneck queue size (in seconds) when

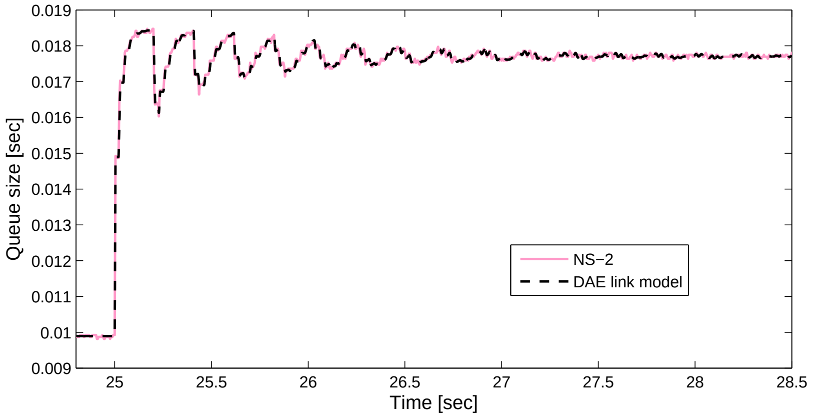


Figure 5.8: Single bottleneck. Two flows with $d_1 = 10\,\text{ms}$, $d_2 = 190\,\text{ms}$. No cross traffic. Flight size of short RTT flow increases.

this scenario is simulated in NS-2, exhibiting significant oscillation in the queue.

The dashed black line shows the continuous time fluid model (3.8), it shows almost perfect agreement with the packet level simulation, even at sub-RTT time scales.

Heterogeneous RTTs and cross traffic

Take the previous scenario, but increase the capacity five times to $c = 750\,\text{Mbit/s}$ and introduce cross traffic sending over the bottleneck with a constant rate $x_c = 600\,\text{Mbit/s}$, i.e., the total available bandwidth available for the window based flows is still 150 Mbit/s. The solid line in Figure 5.9 shows the queue size when this scenario is simulated in NS-2. The dashed line corresponds to the ACK-clocking model (3.8). The accuracy of the ACK-clocking model (3.8) is still remarkable. We also observe that the cross traffic has a smoothing effect on the system.

Linearized model

We have observed that the full ACK-clocking model (3.10) seems to be very accurate. Let us investigate the accuracy of its linear version around the equilibrium.

Two window based flows are sending over a bottleneck link with capacity $c = 100\,\text{Mbit/s}$. There is no non-window based cross traffic, so $x_c = 0$. Initially, $w_1 = 60\,\text{packets}$ and $w_2 = 2000\,\text{packets}$, with packet size $\rho = 1040\,\text{byte}$. Furthermore, $d_1 = 10\,\text{ms}$ and $d_2 = 200\,\text{ms}$. The system is started in equilibrium, and w_1 is increased by 10 at $t = 10$ s, and 300 ms later it is decreased back to 60. The solid line in Figure 5.10 shows the queue size when the system is simulated in NS-2, the dashed line the linear approximation (4.14). The model performs well, so the linear

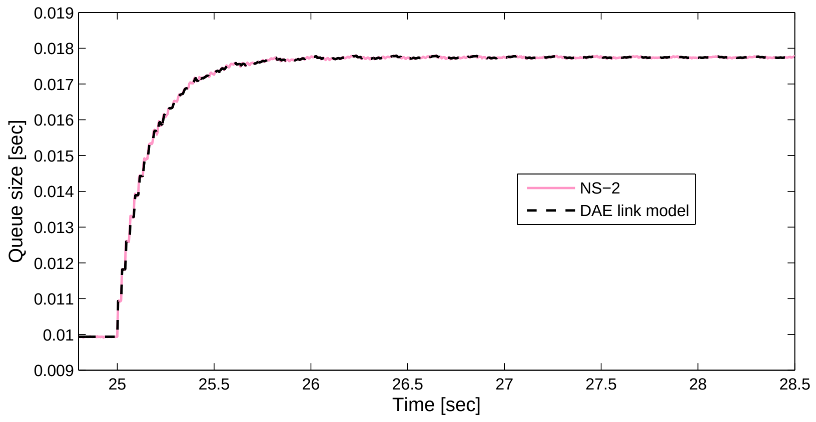


Figure 5.9: Single bottleneck. Two flows with $d_1 = 10\,\mathrm{ms}$, $d_2 = 190\,\mathrm{ms}$. Cross traffic of 80% of link capacity present. Flight size of short RTT flow increases.

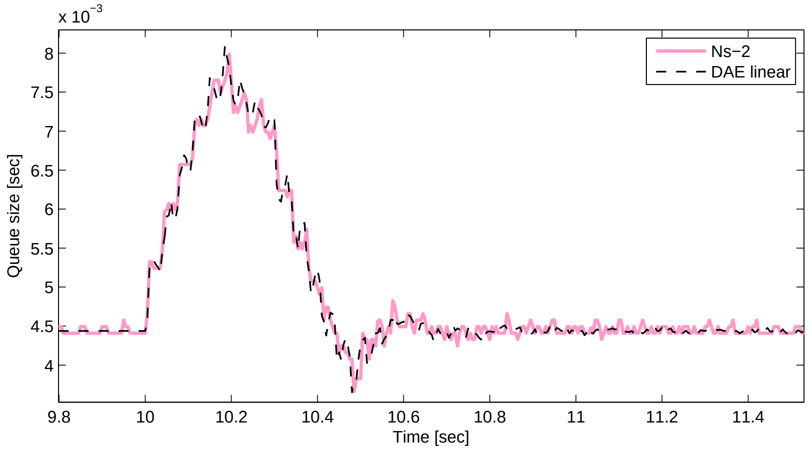


Figure 5.10: Single bottleneck, linearized model. Two flows with delays $d_1 = 10\,\mathrm{ms}$, $d_2 = 200\,\mathrm{ms}$. Short RTT flow has an increased window from 10 s to 10.3 s.

approximation seems valid. In the transfer function version of the linear model, i.e., the model (4.14), exponential functions due to time-delays appears. When simulating the transfer function system in Matlab, Padé approximations of order (17,17) of the exponential functions has been used.

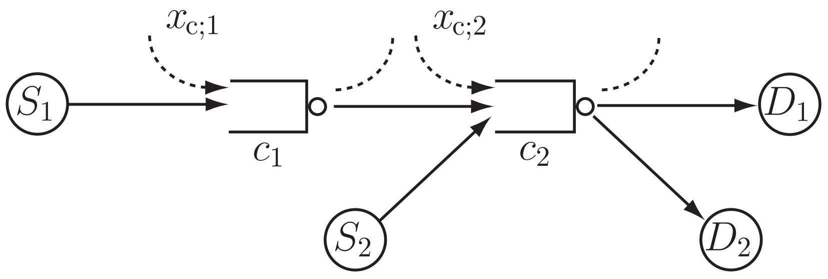


Figure 5.11: Network configuration validation example. The ith source/destination pair is denoted with S_i and D_i.

5.2.2 Multiple link networks

From the previous section it seems clear that the ACK-clocking model is accurate for the single link case. However, it is still open if the performance of the considerable more complex multiple link version is as good. In the validation examples that follow we will investigate this.

The multi-flow multi-link ACK-clocking model (3.10) will here be validated using NS-2 for a scenario with two flows sending over a network of two bottleneck links.

Basic configuration

The configuration is according to Figure 5.11. The first flow utilizes both links, and in the view of this source, the first link is upstream the second link. The second flow is sending over the second link only. Furthermore, there may exist non-window based cross traffic sending over the individual links. For all simulations $c_1 = 80\,\text{Mbit/s}$, $c_2 = 200\,\text{Mbit/s}$, $d_1 = 100\,\text{ms}$, $d_2 = 200\,\text{ms}$, and packet size $\rho = 1040\,\text{byte}$.

Furthermore, the first source is located at the first link and thus $\ell(1) = 1$ (which models zero forward propagation delay), while the second flow is located at a non-bottleneck "link" $\ell(2)$ upstream the second link (modeling forward propagation delay). The configuration is such that $\vec{d}_{1,1,2} = \vec{d}_{2,1,2} = 50$ ms and $\vec{d}_{\ell(2),2,2} = 50$ ms.

The system is perturbed from equilibrium at $t = 15$ s by a positive step change in one of the sources' flight size of magnitude 50 packets. The queue sizes of the simulated DAE model (simulated in Simulink) is compared with NS-2 data.

Case 1: no cross traffic

No traffic except the two window based sources are present, so $x_{c;1} = x_{c;2} = 0$. Furthermore $f_1 = 2100$ and $f_2 = 3900$ packets. In Figure 5.12 the system is perturbed from equilibrium by a step change in the flight size of the first source.

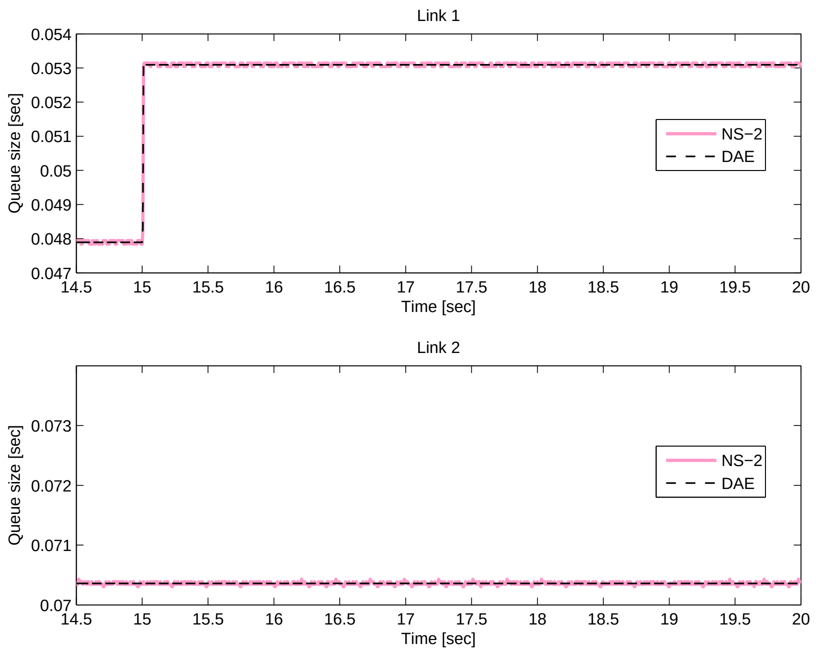


Figure 5.12: Simulation: No cross traffic, step change in Window 1.

We observe that it is only, in the view of the first source, the upstream queue that is affected. This is due to that the traffic traveling from Link 1 to Link 2 is saturated by the capacity of Link 1. This blocking property is captured by the model.

This blocking effect is not observed in Figure 5.13, which shows the result of a change in the second flight size. In that case, both queues are affected even though Link 2 is downstream Link 1 from the point of view of the first source. This is because each source actually is operating in closed loop, and that the ACK rate of the first source is affected by the change in the queue size of Link 2. Moreover we observe that the model fit is very good (in a more detailed view it is observed that the discrepancies are of the magnitude ρ/c_l and hence seem to be due to quantization). The burstiness in the link buffer is captured.

Case 2: cross traffic on Link 1

In this scenario UDP cross traffic is being sent over Link 1 utilizing half the capacity, i.e., $x_{c;1}(t) = c_1/2$. This time initially $f_1 = 2100$ and $f_2 = 3900$ packets. The plot in Figure 5.14 displays the queue sizes when the flight size of the first source is increased.

Figure 5.15 shows the corresponding results when the second source's flight size

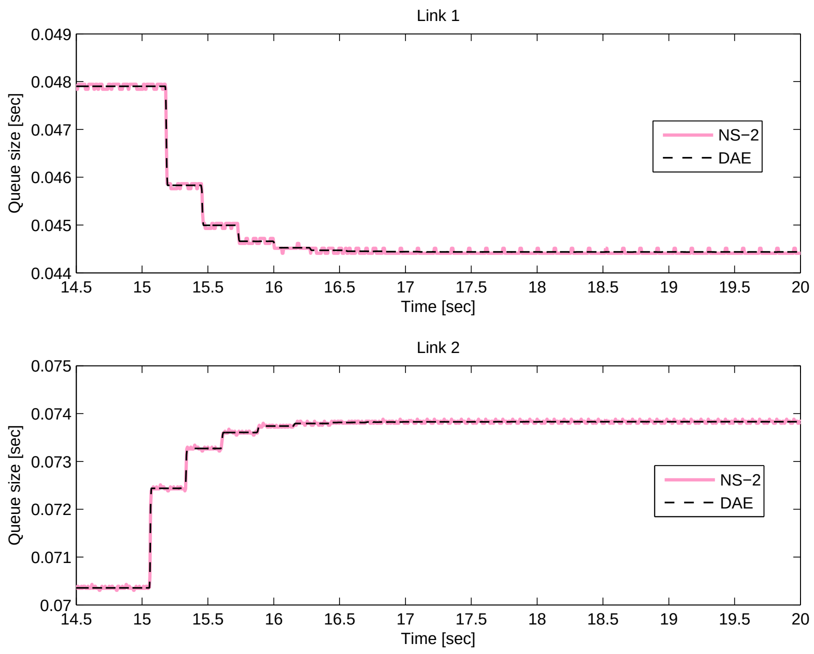


Figure 5.13: Simulation: No cross traffic, step change in Window 2.

is perturbed. Note that the change of the first source affects the second queue size for this case since the flow between the links are not saturated anymore on a shorter time scale.

Case 3: cross traffic on Link 2

In this scenario UDP cross traffic is sending over Link 2 and utilizes half the capacity, i.e., $x_{c;2}(t) = c_2/2$, and initially $f_1 = 2500$ and $f_2 = 600$ packets. Note that in this case, f_2 must be significantly smaller than f_1, if that was not the case, Link 1 would cease to be a bottleneck for Flow 1. The plot in Figure 5.16 displays the queue sizes when the congestion flight size of the first source is increased step-wise. As in the first simulation case, the second queue is not affected. The explanation is analogous. The plot in Figure 5.17 corresponds to the case when the second source's flight size is increased, by 50 packets as usual. Here both queues are affected by the flight size perturbation, and the transient is significant for this case.

Case 4: cross traffic on both links

In this scenario, UDP sources are sending over both links independently, each of them utilizing half the capacity of the link, i.e., $x_{c;1}(t) = c_1/2$ and $x_{c;2}(t) = c_2/2$.

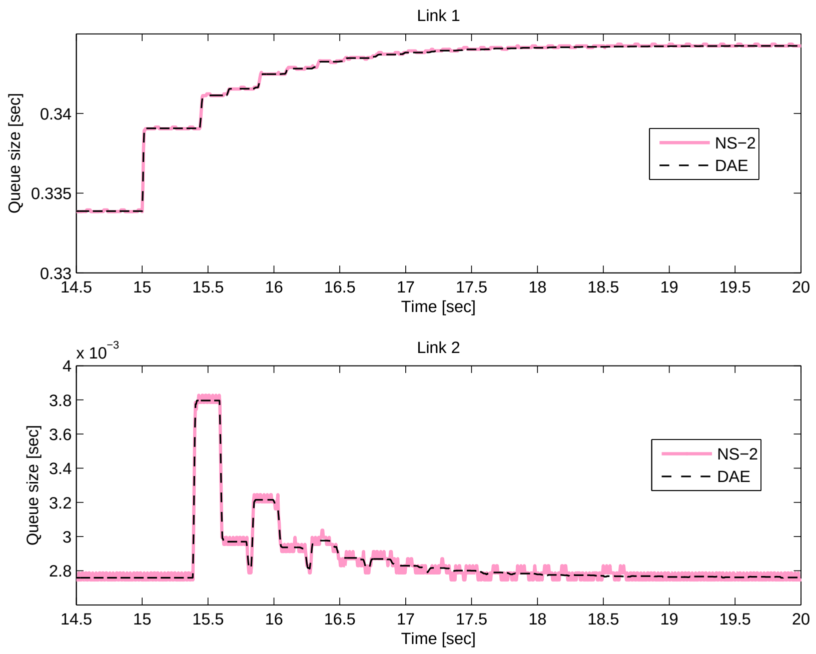


Figure 5.14: Simulation: Cross traffic on Link 1, step change in Window 1.

We also initially have $f_1 = 1000$ and $f_2 = 1500$ packets. The plot in Figure 5.18 displays the queue sizes when the flight size of the first source is increased. Figure 5.19 shows when a step is applied to the second source's flight size.

5.2.3 Testbed results

The foregoing results show that the model provides accurate agreement with NS-2 simulations. However, the question arises whether the model is simply capturing artifacts introduced by NS-2. This section will demonstrate that the model and NS-2 qualitatively match the results obtained by real networking hardware. The reader is referred to Appendix C for details about the experiments as well as complementary multiple link validation experiments.

Single link case

This scenario is analogous to the heterogeneous RTT case in Section 5.2.1, i.e., no cross traffic and step in the flight size of the first source.

Capacity is set to $c = 90\,\text{Mbit/s}$ and the propagation delays of the flows are $d_1 = 3.2$ ms and $d_2 = 116.8$ ms respectively, distributed such that there is no forward delay. Flight sizes are initially set to $f_1 = 50$ packets and $f_2 = 550$ packets

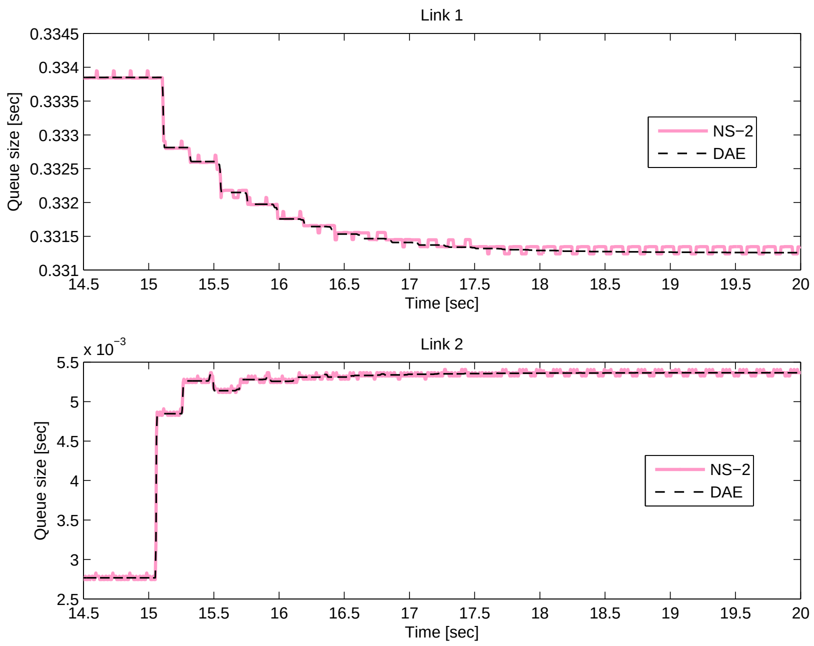


Figure 5.15: Simulation: Cross traffic on Link 1, step change in Window 2.

with packet size $\rho = 1448$ byte. The system is started in equilibrium and at time $t = 25$ s the first source's flight size is increased by 100 to $f_1 = 150$ packets.

The greenish line in Figure 5.20 corresponds to variation in the one way round trip time when the above scenario (with capacity 100 Mbit/s to compensate for lower layer packet overhead) is set up and executed in the testbed. The level of the curve is changed such that it initially matches the equilibrium queue. The pink line is the queue size when the system is simulated in NS-2. Throughout the simulation the flight size is sampled and collected. The dashed black line is the queue size when the DAE model (3.8) is simulated with this time series of the NS-2 flight size as input. We observe a good match between the real testbed data, the NS-2 data, and the model.

Multiple link case

Let us consider a case similar to the one explored in Section 5.2.2, i.e., a scenario with cross traffic on both links utilizing half the capacity on each link. Except for slightly different parameter values, what differs the two configurations is that we have here added a single link flow with flight size 5 packets to Link 1 to be able to measure the delay on this link in the testbed.

Details about the system parameters are given in Section C.6.

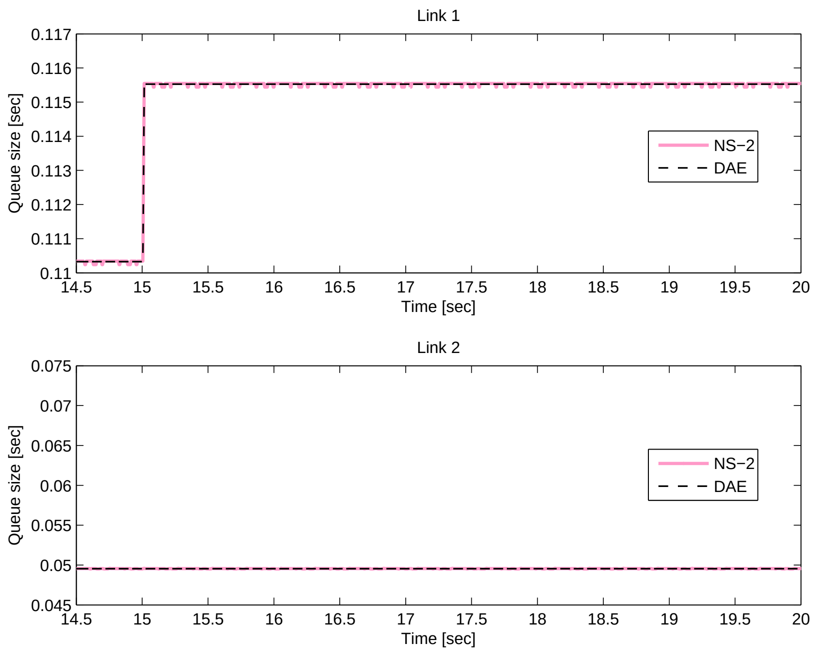


Figure 5.16: Simulation: Cross traffic on Link 2, step change in Window 1.

Figure 5.21 and Figure 5.22 displays when the flight size of the first and second source is increased respectively. The match between the testbed, NS-2 and the model is good.

In summary, the model shows very good agreement with the packet level data derived from both simulations and measurement. It captures sub-RTT effects such as burstiness besides those more long term behaviors.

5.3 Flight size dynamics

We know from the discussion in Section 3.4.3 that the flight size can not be decreased faster than the rate of the received ACKs. This was also the reason for why only positive flight size changes was studied in the validation of the ACK-clocking model in Section 5.2. Next, let us investigate how this inherent traffic shaping, the flight size dynamics, affects the queue size.

The experimental setup is similar to the setup of the ACK-clocking validation where we studied the queue size response to positive step-wise changes in the flight size. However, while we previously were considering the flight size as input signal, we will now apply changes to the window size instead. We do not consider any

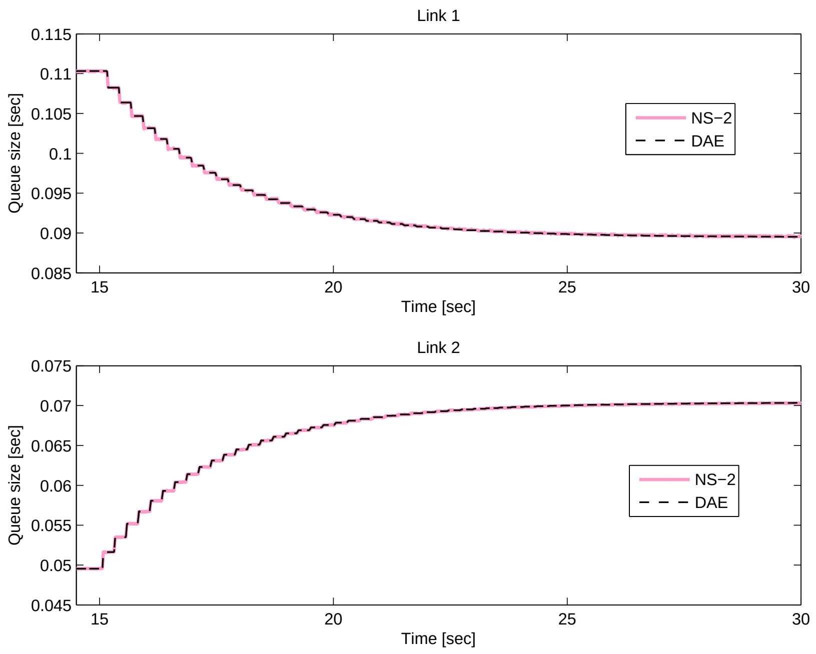


Figure 5.17: Simulation: Cross traffic on Link 2, step change in Window 2.

more advanced traffic shaping than the inherent, therefore only negative changes in the window is of interest. A positive change in the window size (more or less) instantaneously results in an equal increase in the flight size.

Consider a single window based flow sending over a bottleneck link, see Figure 5.5. The link may sometimes also be utilized by non-window based cross traffic. The propagation delay of the window based flow is set to $d = 150\,\mathrm{ms}$ distributed such that there is no forward propagation delay.

Let us study some scenarios where the window size is halved when the system is in equilibrium. The window size is initially set to $w = 500$ packets, and packet size is $\rho = 1040$ bytes.

The plot in Figure 5.23 shows the queue size for such a scenario with capacity $c = 12.5\,\mathrm{Mbit/s}$ and no cross traffic, so $x_c = 0$. The solid line corresponds to the queue when the system is simulated in NS-2. The dashed line corresponds to a simulation of the ACK-clocking model (3.8) and where the flight size is assumed equal to the window size, i.e., neglecting the flight size dynamics. We observe a mismatch between the two simulations. The model is leading compared to the NS-2 simulation. The reason is the inherent traffic shaping, which bounds the rate of change of the flight size. However, modeling this phenomenon with a rate limiter as described in Section 3.4.3, we get a queue according to the dotted line in

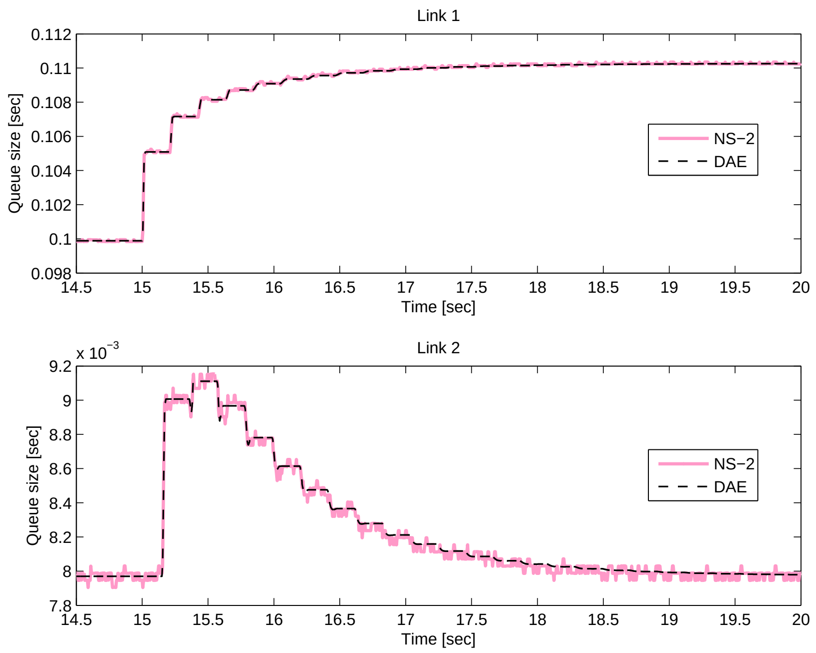


Figure 5.18: Simulation: Cross traffic on both links, step change in Window 1.

Figure 5.23. Now the match is very good.

Doubling the capacity to $c = 25\,\text{Mbit/s}$ and introducing cross traffic with constant rate $x_c = 12.5\,\text{Mbit/s}$, the queue appears as in Figure 5.24. Due to the cross traffic there is now a brief transient. We see from the dashed line that the mismatch from not modeling the flight size dynamics is attenuated after some RTTs. We also observe that at the end of each "stair" in the plot the model is accurate. This is due to that the flight size track the window size within one RTT.

The attenuation of the model error is even more apparent in the transient case of Figure 5.25 which corresponds to a simulation with capacity $c = 125\,\text{Mbit/s}$ and cross traffic $112.5\,\text{Mbit/s}$.

We have so far observed that drastic changes in the window size may introduce a model error on the sub-RTT time scale. This is in line with what we know from Section 4.3. Furthermore, we know from that section that a bound on the error is proportional to the change of the window size, which is intuitive as well. We thus expect a small change in the window to give a smaller model error. The plot in Figure 5.26 corresponds to a scenario with the same configurations as previously but where the window decrease is just 1% (5 packets instead of 250). We can hardly differ between the two models for this case. They both show a very good match.

In summary, modeling the flight size control does improve accuracy for certain

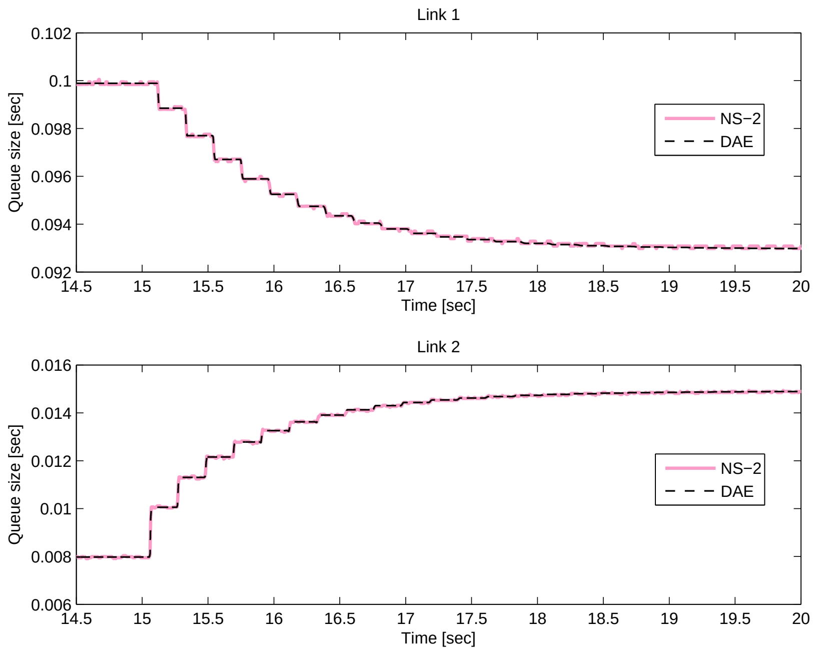


Figure 5.19: Simulation: Cross traffic on both links, step change in Window 2.

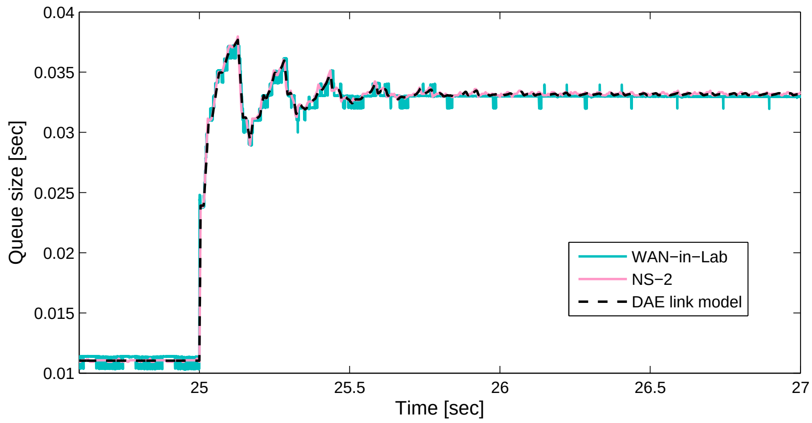


Figure 5.20: Single bottleneck. Two flows with $d_1 = 3.2\,\mathrm{ms}$, and $d_2 = 116.8\,\mathrm{ms}$. Flight size of short RTT flow increases.

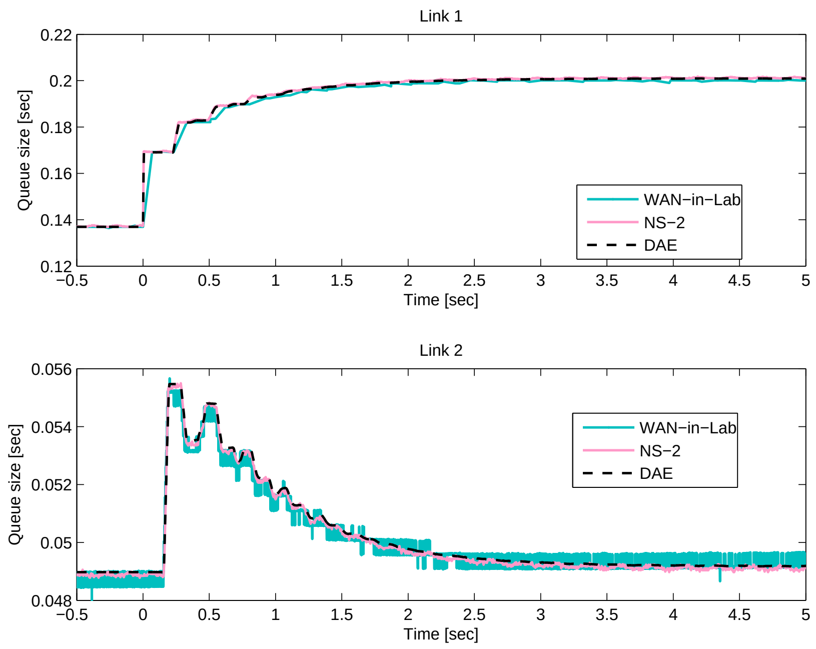


Figure 5.21: Simulation: Cross traffic on both links, step change in Window 1.

scenarios. In particular we have observed that for large violent changes in the window, modeling the flight size may indeed be important. As it seems when not including the flight size control in the modeling the model error in the queue size depends on the relation between the rate of change in the window and the rate of received ACKs. We have also seen that for small changes in the window size, the loss in model accuracy seems minor when ignoring the flight size dynamics. This leads to the conclusion that for protocol simulation it may be wise to model the flight size dynamics, while for, e.g., linear analysis around an equilibrium it is often neglectable.

The more or less perfect match between the queue size of the model that models the flight size and the NS-2 simulation, furthermore, serves as a validation of the flight size dynamics model that was developed in Section 3.4.3.

5.4 FAST TCP

In the previous sections of this chapter we have focused on verifying the accuracy of the ACK-clocking model and the flight size control, dynamics which are generic to all window based congestion control algorithms. In the analysis of FAST TCP in Section 4.4, the stability of the FAST window update mechanism in closed loop

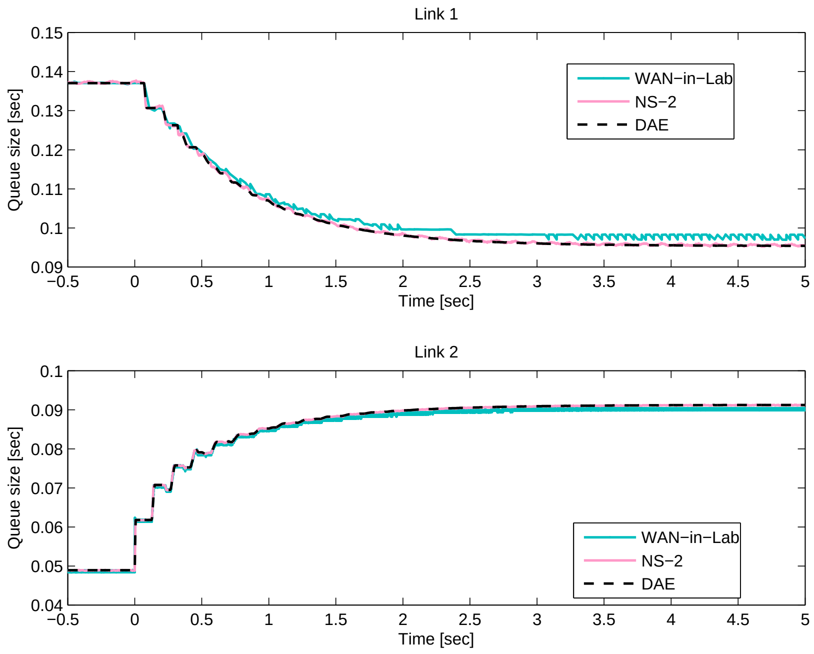


Figure 5.22: Simulation: Cross traffic on both links, step change in Window 2.

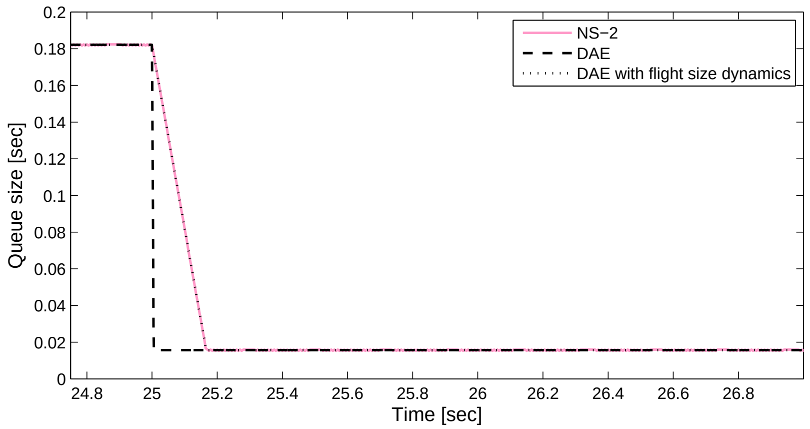


Figure 5.23: Flight size dynamics validation. No cross traffic. Window size halved at $t = 25\,\mathrm{s}$.

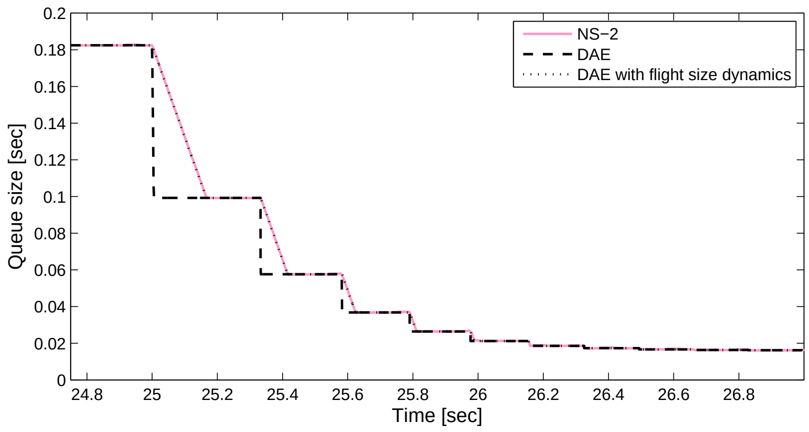


Figure 5.24: Flight size dynamics validation. Half capacity used by cross traffic. Window size halved at $t = 25\,\text{s}$.

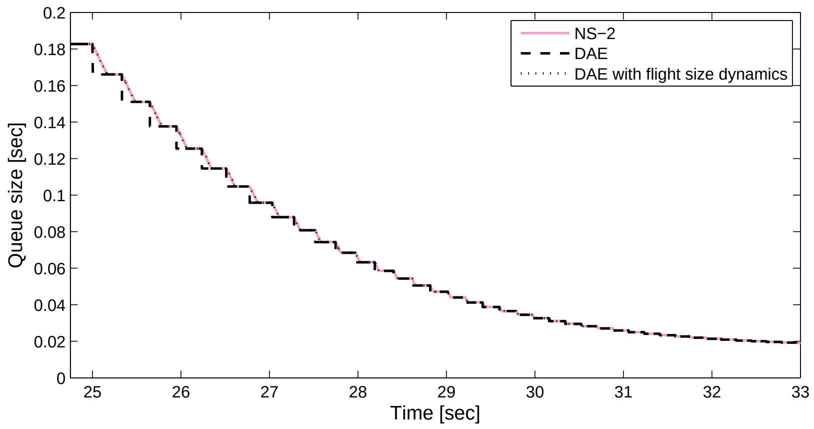


Figure 5.25: Flight size dynamics validation. Cross traffic utilizes 90% of link capacity. Window size halved at $t = 25\,\text{s}$.

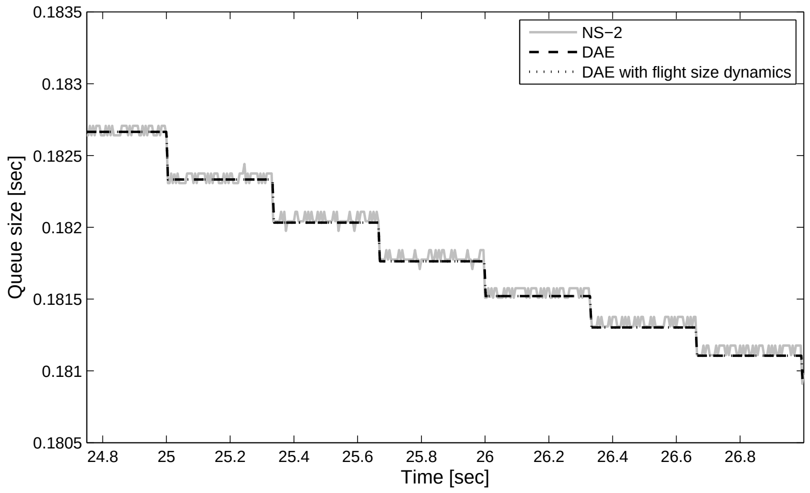


Figure 5.26: Flight size dynamics validation. Cross traffic utilizes 90% of link capacity. Negative change of 1% in the window size at $t = 25\,\mathrm{s}$.

with these more generic parts of the system was studied. The models predicted unstable scenarios that were confirmed with a packet level simulation as well as a testbed experiment, cf., Example 4.4.1 and Example 4.4.2.

The satisfying match between theory and practice serves as a validation of the models used in itself a posteriori. However, for completeness, let us now study the accuracy of the FAST TCP protocol model in more detail.

5.4.1 Experiment setup

The experimental data used in the model validation is generated with NS-2[1]. The data is taken from experiments when the protocol is operating in closed loop. NS-2 is here assumed to correspond to the "true" system.

The basic setting is a single FAST TCP source sending over a single bottleneck link. The multiplicative increase mode of the FAST source is disabled. After the system has converged it is perturbed from the equilibrium by UDP cross traffic starting to send over the link with constant rate of half the capacity. After reaching the new equilibrium the UDP source is turned off.

Time series of the queue size (price) b and the window size w is collected throughout the experiment. Due to that the external perturbation was applied directly to the link, we know from the discussion on closed loop validation in Section 5.1.2 that the data $\{b, w\}$ describes the protocol dynamics and not the queue dynamics.

[1]The default NS-2 version of FAST TCP (Cui and Andrew) bas been slightly modified to match the theoretical specified FAST TCP algorithm described in Section 3.6.

The collected queuing delay data b is used as an input to the protocol model, and the predicted output is compared with the true window data w. The data from the packet level system sometimes slightly disagrees with the theoretical equilibrium predicted by the model, which predicts that exactly α packets is buffered in equilibrium. For such cases the model window ouput has been adjusted to be aligned with the true window size initially before the queue perturbation.

5.4.2 Model validation

The FAST protocol model we want to validate corresponds to the window control and estimator part of the model summarized in Figure 3.20, with window control dynamics given by (4.31) and estimator dynamics by (4.34), and where the window control sampling time is approximated as $h_k(t) \approx d + u_{\hat{q}}(t)$ and the estimator according to $h'_{k'}(t) \approx (d + u_q(t))/u_w(t)$. We only consider the case where $\gamma = 0.5$, $\kappa = 3$ and $\nu = 1/4$, which are default values of the FAST algorithm.

Consider an experiment with bottleneck link capacity $c = 100\,\text{Mbit/s}$, source propagation delay $d = 15\,\text{ms}$ and FAST protocol parameter $\alpha = 200$. UDP cross traffic enters the bottleneck link at time $t = 25\,\text{s}$ and leaves after one second. The solid lines in Figure 5.27 displays data from a NS-2 simulation of this scenario, the upper plot shows the window size and the lower plot the queue size. We see that the queue size is increased as the load on the link is increased, and that the FAST source adapts to this by decreasing the window size accordingly. Due to the small round trip time the converegence is fast (in absolute time).

When the UDP source leaves at $t = 26\,\text{s}$ the system response is analogous. The queue is initially decresed due to the lowered link load and the FAST source increases its window size to compensate for this.

The dashed line in the upper plot shows the window size predicted by the model. The model fit is overall good, but it appears to be a small error in the static gain. It is attributed to the mismatch between the theoretical equilibrium and the equilibrium observed in the packet level experiment.

Increasing the propagation delay to $d = 250\,\text{ms}$ and letting the UDP source send between $t = 25\,\text{s}$ and $t = 40\,\text{s}$, we get results according to Figure 5.28. We observe a slightly more transient system and a much slower response due to the increased delay. The model performance is satisfying for the first transient. For the transient at $40\,\text{s}$ we observe worse behavior. However, studying the queue size in the lower plot we see that it empties. We thus explain the degradation in accuracy with that, even that the multiplicative decrease phase of the protocol has been turned off during the simulation, protocol dynamics not included in the model is active during this period.

Figure 5.29 corresponds to a simulation with parameters $\alpha = 1000$, $c = 50\,\text{Mbit/s}$, $d = 5\,\text{ms}$, and where the UDP cross traffic is turned on at $t = 25\,\text{s}$ and turned of at $t = 28\,\text{s}$. Like before we observe a small error in the static gain.

Increasing the delay to $d = 500\,\text{ms}$ and turning of the UDP source at $35\,\text{s}$ instead we get the results shown in Figure 5.30. Model performance is quantitatively the

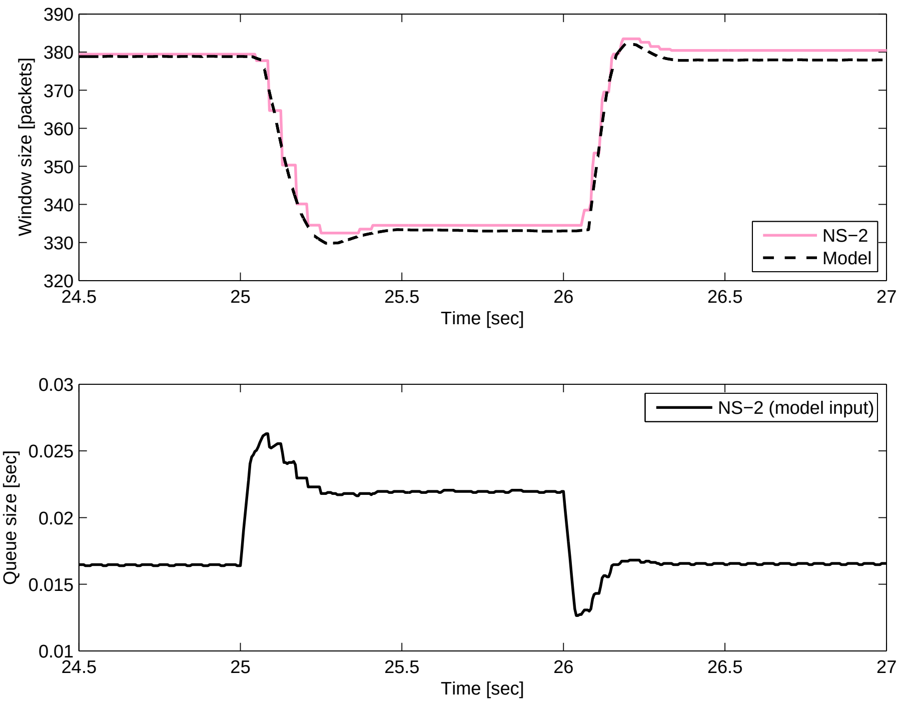


Figure 5.27: FAST TCP model validation. Experiment parameters: $\alpha = 200$, $c = 100\,\text{Mbit/s}$, $d = 15\,\text{ms}$.

same as for previous examples.

In summary we can conclude from the experiments that for the high bandwidth case FAST is designed for, the window and estimator model in Figure 3.20, with window control dynamics given by (4.31) and estimator dynamics by (4.34), and $h_k(t) \approx d + u_{\hat{q}}(t)$ and the estimator according to $h'_{k'}(t) \approx (d + u_q(t))/u_w(t)$, seems quite accurate around equilibrium points. It thus seems suitable to departure from in a linear analysis.

5.5 Summary

This chapter opened with a discussion on how to generate experimental data in closed loop for a window based congestion control system. The insight from the discussion was later on in the chapter used for validating the protocol dynamics of FAST TCP. Next, an extensive validation of the ACK-clocking model derived in Chapter 3 was performed. The model was shown to be very accurate using experimental data from NS-2 as well as a physical testbed. We then investigated

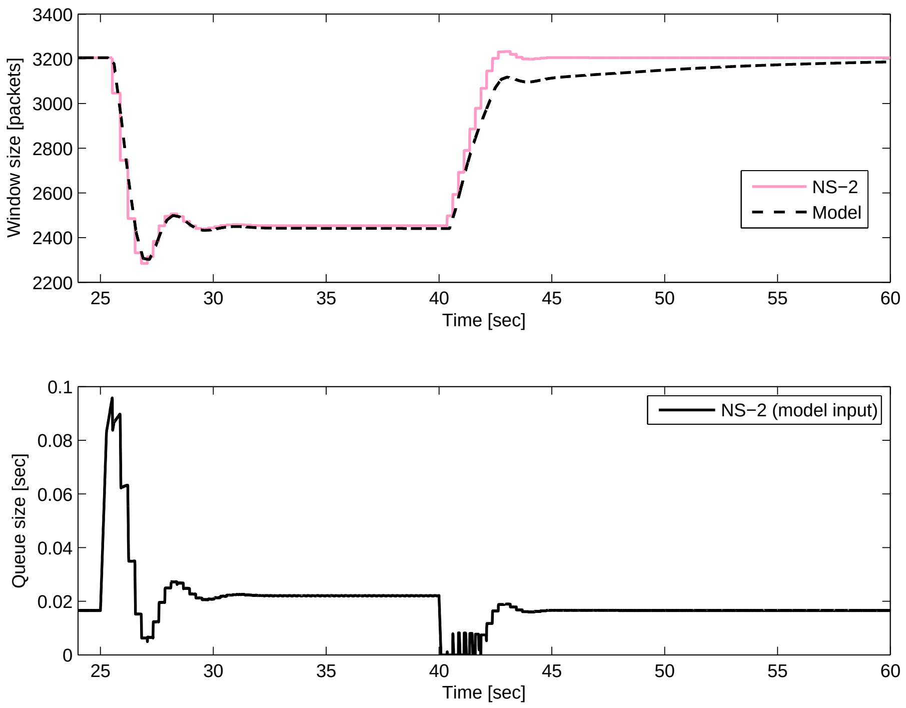


Figure 5.28: FAST TCP model validation. Experiment parameters: $\alpha = 200$, $c = 100\,\mathrm{Mbit/s}$, $d = 250\,\mathrm{ms}$.

the impact of the inherent traffic shaping present in the ACK-clocking mechanism, which we refer to as the flight size control dynamics. We observed a good fit between our model predictions and the experiments. Moreover we saw that for relative small changes in the window size the model error in the queue size when not taking the flight size control into consideration seems negligible. Furthermore, the error was attenuated as the queue converges. At last we validated the simplified continuous time model of FAST TCP that was used for stability analysis in Chapter 4. We saw a reasonable fit of the model, especially for high bandwidth scenarios which FAST TCP is primarily designed for. We observed, however, a slight mismatch in the static gain. The real system actually did not converge to the exact theoretical equilibrium.

5.6 Related work

Considering model identification in closed loop, there is an extensive literature on how to design appropriate excitation and which methods that can be applied, see,

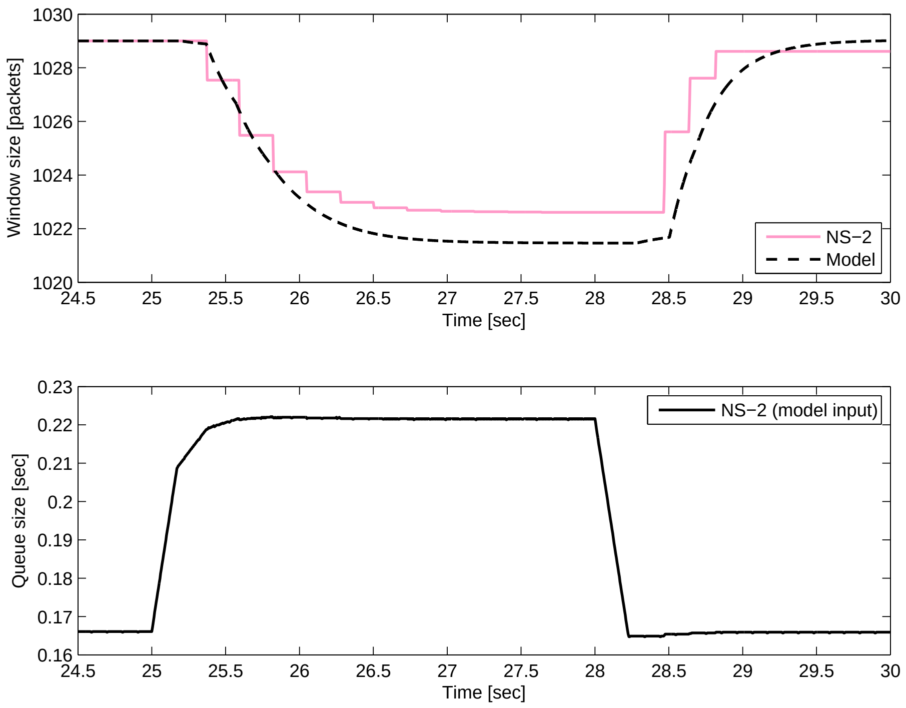


Figure 5.29: FAST TCP model validation. Experiment parameters: $\alpha = 1000$, $c = 50\,\mathrm{Mbit/s}$, $d = 5\,\mathrm{ms}$.

e.g., (Ljung, 1999; Forssell and Ljung, 1999; Van den Hof and Schrama, 1995).

In parts of this work we used the packet level simulator NS-2 (ns2) to generate "true" reference data. Alternative simulators are, e.g., OMNeT++ and OPNET (omnet; opnet). A thoughtful note emphasizing the potential danger on relying to much on network simulation models is (Floyd, 1999). A discussion on the advantages and pitfalls using simulations in the case of evalutaing TCP can be found in Allman and Falk (1999).

Evaluation of TCP is also the topic of the paper (Li *et al.*, 2007), which proposes a series of benchmark tests and compares the performance of five modern high-speed TCPs of which FAST TCP is one. The experimental setup is based on emulation using the Dummynet network emulator (Rizzo, 1997).

In terms of level of abstraction, simulators are on top of emulators. Under the emulation abstraction level comes physical testbeds. Examples of such are the Grid'5000 network (Bolze *et al.*, 2006) and the WAN-in-Lab facility (Lee *et al.*, 2007a). The latter was in this thesis used to validate the ACK-clocking model. The validity of using NS-2 simulations as reference for the type of experiments

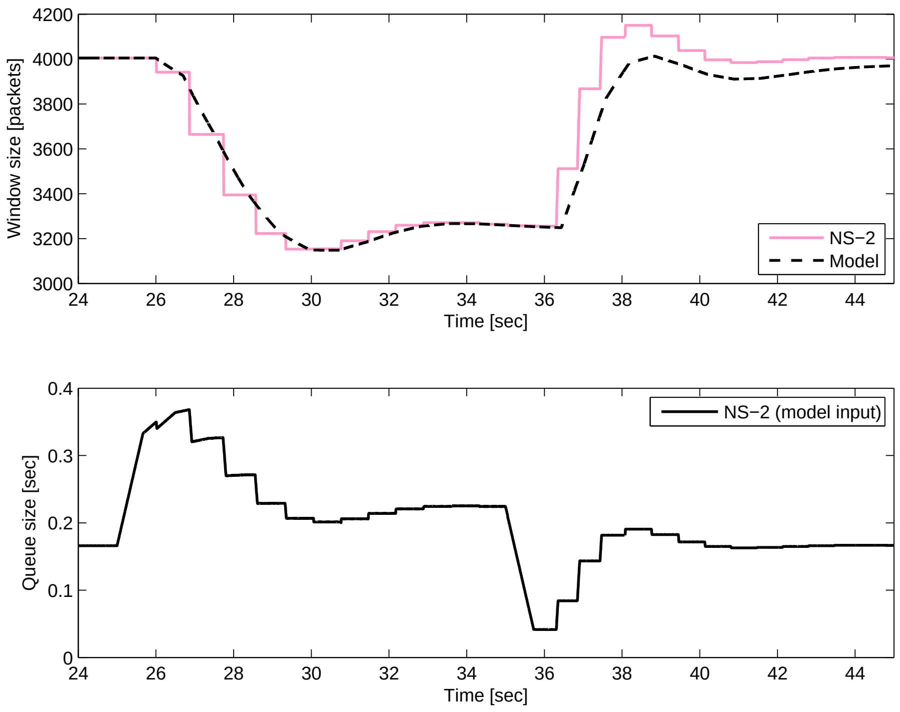


Figure 5.30: FAST TCP model validation. Experiment parameters: $\alpha = 1000$, $c = 50\,\mathrm{Mbit/s}$, $d = 500\,\mathrm{ms}$.

that were executed was confirmed simultaneously. The ultimate testing ground for transport layer protocols are full production networks as, e.g., the Ultralight network (Newman *et al.*, 2006). However they are sensitive for testing failure modes that may disrupt other traffic.

Chapter 6

Conclusions and future work

TO conclude the thesis we will now summarize the results presented in the previous chapters and suggest some possible future extensions to this work. But first let us recapitulate what we have done. In Chapter 3 we developed a fluid flow model of a network utilized by window based congestion control protocols using queuing delay as network congestion notification. In particular we derived a detailed model of the ACK-clocking mechanism. In Chapter 4 we analyzed the properties of the ACK-clocking mechanism and furthermore the stability of FAST TCP. Chapter 5 was devoted to validation of the models derived in Chapter 3 and used for analysis in Chapter 4.

6.1 Conclusions

The main contribution of this thesis is a detailed dynamical characterization of the ACK-clocking mechanism, generic to all window based congestion control protocols. The model was derived by viewing the window based system as decomposed into two cascaded feedback loops, cf., Figure 3.4, where the ACK-clocking mechanism and the network constitutes the inner loop, which is to be controlled by the window control in the outer loop.

The key idea in the modeling is to consider the instantaneous rate a window based sender causes in each queue in the network, in contrast to only explicitly model the sender sending rate which is customary in previous work. By relating this rate with the flight size (and, thus, the window implicitly) results in a fundamental integral equation. The model is very accurate even at sub-RTT time scales. It even captures traffic burstiness patterns present in the underlying packet level system and could thus be used for analyzing, e.g., loss synchronization of loss-based TCPs.

For the single bottleneck case, individual rates may oscillate in steady-state, while the total rate into the link and the queue remain constant. This “self clocking” phenomena is due to that the throughput of a link is constrained by the capacity. This implies that the total sum of the rates with which senders’ receive ACKs, and thus the sum of the sending rates, must be equal to the bottleneck capacity in steady-state. Due to similar reasons, a bottleneck link in a multiple link network has a smoothing effect on the arrival rate of downstream links. Changes in senders' sending rates may not affect links downstream the first bottleneck in the path. These "self clocking" and "blocking" effects are indeed captured by the model. We observe that any feasible equilibrium of the ACK-clocking system is unique if it exists, and that the queue dynamics is stable for small perturbations around this equilibrium. It is furthermore realized that the amount of non-window based cross traffic significantly affects the time constants of the system—the more cross traffic the slower dynamics.

The ACK-clocking model presented in this work predicts previously unknown dynamical behavior. As an example, the ACK-clocking mechanism becomes significantly oscillatory when flows' RTTs are wide apart. This is not captured by existing models that predicts a smooth convergence. To investigate the impact of this on the macroscopic performance of window based protocols using queuing delay as control signal, we developed a detailed fluid model of a generic protocol taking, e.g., estimator dynamics and sampling effects into account. The framework facilitates modeling of a quite large class of endpoint algorithms. Modeling FAST TCP using this framework and analyzing its behavior leads to a new observation—FAST TCP becomes unstable as the difference between flows' RTT increase. While this result is interesting as such, perhaps the most important contribution is the developed stability analysis technique. It is also demonstrated how the detailed models can be simplified to meet the accuracy needed for the intended use. Maybe the most intriguing conclusion we make from the analysis is that the gain of the ACK-clocking mechanism dramatically increases with the RTT heterogeneity for the case of an equal resource allocation policy. Since this strongly affects the stability properties of the system, this is alarming for all window based congestion control protocols striving towards proportional fairness. Notice that this caution is in line with what we observed for the case of FAST TCP.

Many of the theoretical results that appeared in the thesis were verified with packet level experiments using the NS-2 simulator and the Wan-in-Lab physical testbed. Apart from the quantification of the accuracy of the models, a very important observation is that NS-2 to a large extent reflects the behavior of a real system for the type of examples considered. An issue constantly debated in this type of work.

6.2 Future work

While we by a more detailed fluid flow modeling approach than customary have revealed some interesting properties concerning the dynamics of window based congestion control, there are still plenty of work to pursue.

By analyzing FAST TCP we have seen the potential consequences the ACK-clocking dynamics may have on the outer loop, that is the window control, for protocols using delay as congestion notification. The observation that the gain dramatically increases as the RTT heterogeneity among flows is increased leads us to conjecture that there is an inherent performance limitation in window based congestion control striving towards a proportionally fair resource allocation. The flow rates are coupled to each other through the ACK-clocking mechanism. Thus each flow is subject to feedback on time scales proportional to the other flows' RTT. While it is quite straightforward for a flow to scale down the gain proportional to its own RTT, like FAST TCP does, it is not obvious how to compensate for the feedback on the other time scales without communicating the RTTs between endpoint flows explicitly. Standard TCP solves this issue by implicitly adapting the gain to other flows' RTT by a resource allocation inversely proportional to the RTT (the gain of a flow is proportional to the sending rate). Is it possible to achieve the same effect by utilizing other implicit information? A future study answering this question is of interest. In the perspective that the partial objective of many modern TCP proposals is to avoid RTT biased steady-state rates and the trend that TCP proposals responds to short time scale dynamics of queue sizes, exposing the properties of the ACK-clocking for a proportional resource allocation is relevant.

The accuracy in the ACK-clocking model allows for the analysis of sub-RTT phenomena. This implies that it has the potential to analyze, e.g., loss synchronization which has a significant impact on the fairness of loss-based congestion control. To be able to do so, however, the model needs to be extended to handle packet loss. Some initial steps in this direction is taken in (Tang *et al.*, 2008). Nevertheless, we remark that fluid modeling and analysis of loss-based TCP in a drop-tail network environment still is an open problem.

While it certainly is of interest to extend the models presented here to handle more complex scenarios, like, e.g., packet loss, it is also of interest to go in the other direction and investigate appropriate model simplifications. The multiple link ACK-clocking model, e.g., is not very tractable to any analysis and reasonable approximations would be of great value.

In this thesis we have focused on window based congestion control, originally motivated by the "conservation of packets" principle by Jacobson (1988). Although this is a sound idea, we have seen that letting senders maintain a window imposes complex system dynamics that is not well understood. It is desirable that future designs are not only guided by performance objectives, but also directed by analysis tractability constraints. An alternative to window based congestion control may be to further explore algorithms which explicitly controls the transmission rate.

Appendix A

Quadrature approximations of the ACK-clocking model

At a first sight *numerical quadrature* techniques seem suitable when approximating the integral equation defining the instantaneous rates x_n in the ACK-clocking model (3.8),

$$\begin{aligned}
&\dot{b}(t) - \frac{1}{c}\left(\sum_{n=1}^{N} x_n(t) + x_c(t) - c\right) = 0,\\
&\int_t^{t+\tau_n(t)} x_n(s)\,ds - f_n(t+\tau_n(t)) = 0, \quad n = 1,\ldots,N.
\end{aligned}$$

Numerical quadrature is a well developed technology in numerical integration and numerical solutions to integral equations. Quadrature rules are based on polynomial interpolation. The integrand function is sampled at a number of points, the polynomial that interpolates the function at those points is determined, and the integral of the interpolant is then taken as an approximation of the integral of the original function. Applying such techniques, suitable order of model accuracy can be chosen without increasing the order of the approximating delay DAE. The delay DAE will remain first order like the original system while the model complexity is kept in the more or less complex recursive update of the rate. We will now demonstrate this, and we will see that we may get non-causal as well as unstable models. Furthermore, even the simplest rules yields approximative models that are of higher complexity in comparison with the original system model on accumulated data form (3.9). The use of approximative models based on this type of approximation thus seems to be limited.

A.1 Model approximations

Let us assume small intervals of integration τ_n and sufficiently smooth rates x_n, and start with some simple one-point quadrature rules. Recall (4.19),

$$H_n(t,z) = \int_t^z x_n(s)ds - f_n(z),$$

and (4.20),

$$H_n(t, t+\tau_n(t)) = 0.$$

Applying the right-side *rectangle rule* to (4.20) gives

$$0 = H_n(t, t+\tau(t)) = \tau_n(t)x_n(t+\tau_n(t)) + \mathcal{O}(\tau_n^2) - f_n(t+\tau_n(t))$$

and thus an approximative rate model, with error term $\mathcal{O}(\tau)$, is

$$x_n(t) \approx \frac{f_n(t)}{\tau_n(t-\tau_n(\tilde{t}))}, \tag{A.1}$$

where $\tilde{t}$ satisfies $\tilde{t} + \tau_n(\tilde{t}) = t$. A more accurate version of the rectangle rule is the *midpoint rule*. Applying this rule instead yields

$$0 = H_n(t, t+\tau(t)) = \tau_n(t)x_n(t+\tau_n(t)/2) + \mathcal{O}(\tau_n^3) - f_n(t+\tau_n(t)),$$

and subsequently

$$x_n(t+\tau_n(t)/2) = \frac{f_n(t+\tau_n(t))}{\tau_n(t)} + \mathcal{O}(\tau_n^2). \tag{A.2}$$

We observe that despite that this model is one order more accurate than (A.1) and of the same level of complexity, it is still not tractable due to that it is non-causal, i.e., the rate $x(t)$ at t is a function of future values of the flight size $f(\hat{t})$, $\hat{t} > t$. This highlights that rules such as the midpoint rule that do not evaluate the endpoint of the interval will yield non-causal models with limited usage, with $x(t)$ dependent on a future value of the flight size f, clearly not consistent with physics.

To increase the model accuracy we need to apply more accurate numerical quadrature that interpolates the function values at several points in the interval. Such simplifications gives the approximate rate $x_n(t)$ in terms of recursive update rules. For example, the *trapezoidal rule*, which is a two point quadrature rule interpolating the function values at the two endpoints, gives a recursive rule

$$x_n(t+\tau_n(t)) \approx \frac{2f_n(t+\tau_n(t))}{\tau_n(t)} - x_n(t), \tag{A.3}$$

with error term of order $\mathcal{O}(\tau_n^2)$. An even better $\mathcal{O}(\tau_n^4)$ approximation

$$x_n(t+\tau_n(t)) \approx \frac{6f_n(t+\tau_n(t))}{\tau_n(t)} - 4x_n(t+\tau_n(t)/2) - x_n(t) \tag{A.4}$$

is achieved by applying a three-point quadrature rule known as the *Simpson's* rule.

If the integration interval, i.e., the round trip time $\tau_n(t)$, is not sufficiently small and the rates $x_n(t)$ are varying significantly within an interval it is often suitable to subdivide the original interval into subintervals and then apply a lower-order quadrature rule in each interval.

The approximations based on the rectangle, trapezoidal and Simpson's rule are now compared in a validation experiment.

Example A.1.1. *The scenario is identical to the one described in Example 4.2.3. The system is simulated in NS-2 and compared with the models based on the three quadrature approximations explicitly discussed here.*

The solid grey line in Figure A.1 is the queue size when the system is simulated in NS-2, the solid black line the model based on the rectangle rule rate (A.1), the dotted black line the model based on the trapezoidal rule rate (A.3), and the dashed black line the model based on the Simpson's rule rate (A.4).

In the example we see that the simplest rectangle rule approximation seems crude compared to the previously investigated models based on Taylor expansion and Padé approximation, cf., Section 4.2.4. More remarkably we also observe that the two higher order quadrature approximations seem unstable. In Appendix A.2 this is studied in more detail and it is confirmed that indeed the equilibrium is unstable for these two cases.

A.2 Stability analysis

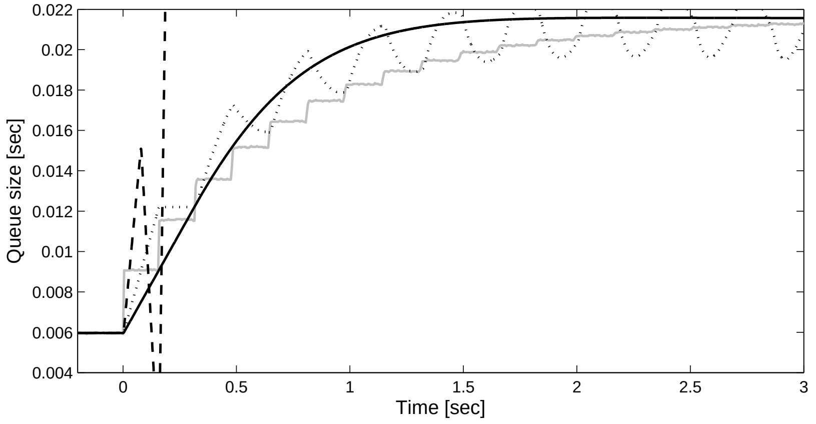
A.2.1 Trapezoidal rule

The approximative model based on the rectangle rule rate model for a single source sending over a single bottleneck is given by (3.4) and (A.3), i.e.,

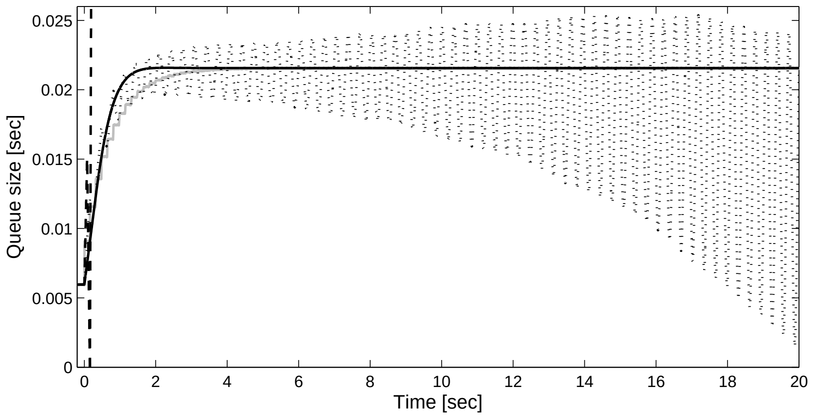
$$\begin{aligned} \dot{b}(t) - \frac{1}{c}\left(x(t) + x_c(t) - c\right) &= 0, \\ x(t+\tau(t)) + x(t) - \frac{2f(t+\tau(t))}{\tau(t)} &= 0. \end{aligned}$$

We will now investigate if the equilibrium is stable. Recall that $\tau(t) = d + b(t)$ and assume for simplicity static cross traffic x_c. Now linearization under the standard assumptions stated in Section 4.2.1 yields

$$\begin{aligned} \delta\dot{b}(t) - \frac{\delta x(t)}{c} &= 0, \\ \delta x(t+\tau) + \delta x(t) + \frac{2x}{\tau}\delta b(t) - \frac{2}{\tau}\delta f(t+\tau) &= 0. \end{aligned}$$



(a) Transient view.



(b) Static view.

Figure A.1: Validation experiment. Solid grey line: NS-2 simulation. Solid black line: rectangle rule rate model (A.1). Dotted black line: trapezoidal rule rate model (A.3). Dashed black line: Simpson's rule rate model (A.4).

Taking the Laplace transform gives

$$\Delta B(s) = \frac{\Delta X(s)}{cs},$$
$$\Delta X(s) = -\frac{2xe^{-\tau s}}{\tau\left(1+e^{-\tau s}\right)}\Delta B(s) + \frac{2}{\tau\left(1+e^{-\tau s}\right)}\Delta F(s),$$

and thus the transfer function from the flight window f to the queue b is given by

$$G_{bf}^{tra}(s) = \frac{2}{c\tau\left(1 + e^{-\tau s}\right)s + 2xe^{-\tau s}}. \tag{A.5}$$

Theorem A.2.1. *The transfer function $G_{bf}^{tra}(s)$ defined by (A.5) has poles in the right hand side complex half plane and is thus unstable.*

Proof. First we note that $G_{bf}^{tra}(s)$ has no zeros and thus we rule out potential pole-zero cancellations. Rewrite the transfer function as

$$G_{bf}^{tra}(s) = \frac{2}{c\tau s} \cdot \frac{1}{1 + e^{-\tau s}\left(1 + 2x/(c\tau s)\right)} = \frac{2}{c\tau s} \cdot \frac{1}{1 + G_0(s)}$$

where

$$G_0(s) = e^{-\tau s}\left(1 + 2x/(c\tau s)\right).$$

We can now study if $G_{bf}^{tra}(s)$ has poles in the right complex half plane by using the Nyquist criterion on $G_0(s)$. We have

$$\begin{aligned} |G_0(j\omega)| &= \frac{\sqrt{\omega^2 + \left(2x/(\tau c)\right)^2}}{\omega} > 1, \\ \arg G_0(j\omega) &= -\tau\omega + \arctan(\omega\tau c/(2x)) - \pi/2. \end{aligned}$$

Since $\arg G_0(j\omega) < 0$ for all $\omega > 0$ and $\arg G_0(j\omega) \to -\infty$ as $\omega \to \infty$ the Nyquist curve of G_0 must encircle $-1 + j0$. By observing that $G_0(s)$ lacks unstable poles except for a single pole in the origin itself we can conclude that $1/(1 + G_0(s))$ is unstable, and thus that $G_{bf}^{tra}(s)$ is unstable. □

A.2.2 Simpson's rule

The approximate model is for this case given by (3.4) in combination with (A.4), i.e.,

$$\begin{aligned} \dot{b}(t) - \frac{1}{c}\left(x(t) + x_c(t) - c\right) &= 0, \\ x(t + \tau(t)) + 4x(t + \tau(t)/2) + x(t) - \frac{6f(t + \tau(t))}{\tau(t)} &= 0. \end{aligned}$$

Under the same assumptions as in Section A.2.1 we linearize the model, yielding

$$\begin{aligned} \delta b(t) - \frac{\delta x(t)}{c} &= 0, \\ \delta x(t + \tau) + \delta 4x(t + \tau/2) + \delta x(t) - \frac{6}{\tau}\delta f(t + \tau) + \frac{6x}{\tau}\delta b(t) &= 0. \end{aligned}$$

By the Laplace transform we have

$$\Delta B(s) = \frac{1}{c}\Delta X(s),$$
$$\Delta X(s) = \frac{6/\tau}{1+4e^{-s\tau/2}+e^{-s\tau}}\left(-xe^{-s\tau}\Delta B(s)+\Delta F(s)\right),$$

and the thus the transfer function from the flight window to the queue is

$$G_{bf}^{sim}(s) = \frac{6/\tau}{c\left(1+4e^{-s\tau/2}+e^{-s\tau}\right)s+6xe^{-s\tau}/\tau} \tag{A.6}$$

for this case. Let us now study the stability of this transfer function.

We first note that $G_{bf}^{sim}(s)$ has no zeros so pole-zero cancellations do not occur. Now, write the transfer function as

$$G_{bf}^{sim}(s) = \frac{6}{c\tau s}\cdot\frac{1}{1+e^{-s\tau}\left(1+4e^{s\tau/2}+6x/(c\tau s)\right)} = \frac{6}{c\tau s}\cdot\frac{1}{1+G_0(s\tau)}$$

where

$$G_0(s) = e^{-s}\left(1+4e^{s/2}+6x/(cs)\right).$$

Since $G_0(s)$ has no unstable poles outside the origin we have by the Nyquist criterion that $G_{bf}^{sim}(s)$ has poles in the right complex half plane if and only if the Nyquist curve of $G_0(s)$ encircles $-1+j0$. We have

$$\lim_{\omega\to\infty}\arg G_0(j\omega) = \lim_{\omega\to\infty}\left\{-\omega-\pi/2+\arctan\left(\frac{\omega(1+4\cos(\omega/2))}{6x/c-\omega\sin(\omega/2)}\right)\right\}\to-\infty.$$

Thus, if $|G_0(j\omega)|>1$ for all ω we must encircle $-1+j0$. This seems to be the case according to Figure A.2. We are therefore confident enough to pose the following conjecture.

Conjecture A.2.2. *The transfer function $G_{bf}^{sim}(s)$ defined by (A.6) has poles in the right hand side complex half plane and is thus unstable.*

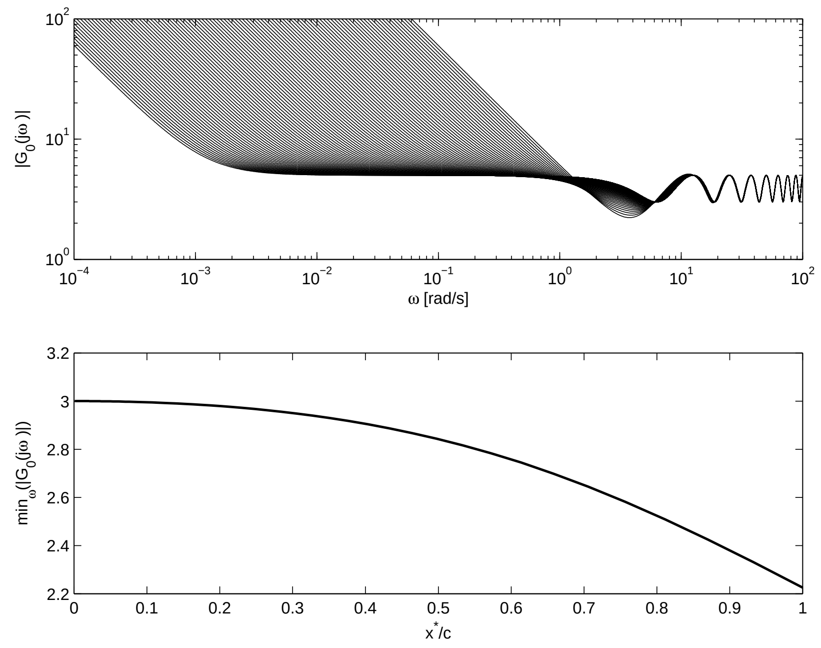


Figure A.2: Analyzing the stability of Simpson's rule rate model. Upper plot: The gain of $G_0(j\omega)$, the different curves corresponds to different (logarithmically spaced) values of $x/c \in (0, 1]$. Lower plot: The minimum gain of $G_0(j\omega)$ as a function of the parameter x/c.

Appendix B

Proof of Theorem 4.4.4

The results appearing in this appendix are taken from Tang *et al.* (2008).

First, let

$$\lambda = \tau_1/(\tau_1 + \tau_2).$$

Let $\omega(\tau_1, \lambda, \beta)$ be the solution to

$$\omega\tau_2 = 2\pi - \omega\tau_1 - \beta,$$

with $\omega\tau_1 \in (0, 2\pi), \omega\tau_2 \in (0, 2\pi)$ and $\beta \in (0, \omega\tau_1)$. Let

$$\begin{aligned} L_\lambda(\beta) &= L(j\omega(\tau_1, \lambda, \beta)) \\ &= \gamma \frac{\dfrac{e^{-j\omega\tau_1}}{j\omega\tau_1}(1 - e^{j(\omega\tau_1+\beta)}) + \dfrac{e^{j\omega\tau_1}e^{j\beta}(1 - e^{-j\omega\tau_1})}{j(2\pi - \omega\tau_1 - \beta)}}{2 - e^{j\omega\tau_1}e^{j\beta} - e^{-j\omega\tau_1}} \end{aligned} \tag{B.1}$$

where each ω in (B.1) refers to $\omega(\tau_1, \lambda, \beta)$.

The proof of the theorem is based on the following lemma.

Lemma B.1. *Let $\omega^* = 2\pi/(\tau_1 + \tau_2)$. Then (4.58) satisfies*

$$\mathrm{Im}(L(j\omega^*)) > 0. \tag{B.2}$$

Let $\beta = (2\pi\lambda)^3$. For $0 < \lambda < 1/(2\pi)^2$, (B.1) satisfies

$$\mathrm{Im}(L_\lambda(\beta)) < 0 \tag{B.3}$$

and for all $\omega \in [\omega^* - \beta/\tau_1, \omega^*]$, (4.58) satisfies

$$|L(j\omega)| \geq \frac{\gamma}{411\lambda^2}. \tag{B.4}$$

For all $\omega > 0$, the general case (4.14), and hence (4.58), satisfies

$$\frac{d}{d\omega}\arg(L(j\omega)) \leq 0. \tag{B.5}$$

Proof. By Lemma 8–11 in Tang *et al.* (2008). $\square$

We are now ready to prove Theorem 4.4.4.

Proof of Theorem 4.4.4. By (B.5) and the Nyquist criterion, (4.58) is unstable if and only if $L(j\omega) \in (-\infty, -1)$ for some ω, since low frequency behavior is as in Paganini *et al.* (2005).

Let $\tilde{\tau} = \max(\sqrt{411/\gamma}, (2\pi)^2)$. For any $\tau_2 > \tilde{\tau}$, the right hand side of (B.4) exceeds 1, and (B.2) and (B.3) of Lemma B.1 hold. By (B.2) and (B.3), $L(j\omega)$ crosses the real axis for some $\omega \in [\omega^* - \beta/\tau_1, \omega^*]$, and by (B.5), this crossing must be clockwise and must be a crossing of the negative real axis. By (B.4) of Lemma B.1, this crossing must be to the left of $-1 + j0$, proving the instability. $\square$

Appendix C

ACK-clocking validation

This appendix consists of additional ACK-clocking validation examples using the WAN-in-Lab testbed.

C.1 Testbed description

The testbed used for this validation consisted of three Cisco 7609 routers connected by 2.5 Gbit/s OC48 links, with spools of fiber used to implement delays (the cover picture of the thesis shows a few of them). Traffic shaping was applied to the OC48 interfaces to obtain the desired link capacities.

Three end-systems were attached to each router by short gigabit Ethernet links. These systems ran Linux 2.6.23, patched to improve the speed of SACK processing. The constant window sizes were obtained using a custom kernel module which sets TCP's window to a constant value, controllable in real time using a `sysctl` variable. Both TCP and UDP traffic was generated using iperf.

The routers used for this experiment do not provide accurate queuing delay measurements. In order to obtain queuing delays, one-way packet delays were measured by matching packets from traces generated by `tcpdump`. The end-systems' clocks were synchronized by calling `ntpdate` immediately prior to starting the dump, but the network time protocol (ntp) daemon was disabled during the experiment to avoid jumps in the measured delays.

To avoid the need for an absolute time reference when performing quantitative comparisons with the model, time series obtained from the testbed were manually aligned so that the initial transient is aligned with the corresponding transient in the simulation. Similarly, the measured packet delays were offset so that the delay before the transient matched the simulated queuing delay at that point. Residual clock skew can be seen in some of the experimental results, in the form of a slight apparent drift in the queuing delays. As an extension to these experiments precise synchronization techniques should be employed (Pásztor and Veitch, 2002).

Testbed capacities include about 10% overhead. Thus, a link described as

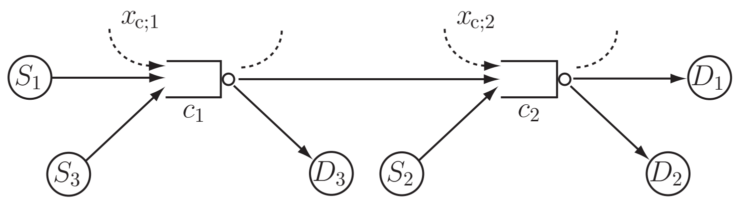


Figure C.1: Network configuration for the validation example. The ith sender destination pair is denoted with S_i and D_i.

200 Mbit/s would be able to carry 180 Mbit/s of iperf traffic. The link capacities and amount of cross traffic in simulation and analytic results was tuned to 90% of the testbed values to match the testbed results.

C.2 Network configuration

Recall that queue measurements in the testbed are performed by capturing TCP packets on end-systems before and after the bottleneck link. In the topology of Figure 5.11, no TCP traffic terminates at the end of Link 1. To measure the delay on this link, a single link flow with window 5 packets was added to Link 1. The configuration is thus as shown in Figure C.1.

Despite its low rate, for some cases, this flow contributes to give qualitatively different results compared with the two flow case studied in Section 5.2.2.

The physical layer link capacities were set to 80 and 200 Mbit/s, giving transport layer capacities $c_1 = 72\,\text{Mbit/s}$, $c_2 = 180\,\text{Mbit/s}$. Also $d_1 = 40\,\text{ms}$, $d_2 = 80\,\text{ms}$, and packets had a transport layer payload of $\rho = 1448\,\text{byte}$. Note that shorter delays were used in this case because of the limited amount of fiber delay available in the testbed. The forward delay from each source to the first bottleneck it encountered is approximately zero.

To enable the main transients to be observed over the jitter, the step change in window sizes was 200 packets.

The flight size of the third flow is kept constant $f_3 = 5$ packets in all experiments.

C.3 Case 1: no cross traffic

No traffic except the two window based sources are present, so $x_{c;1} = x_{c;2} = 0$. Furthermore $f_1 = 1600$ and $f_2 = 1200$ packets. In Figure C.2 the system is perturbed from equilibrium by a step change in the flight size of the first source. We observe that it is only, in the view of the first source, the upstream queue that is affected. This is due to that the traffic traveling from Link 1 to Link 2 is saturated by the capacity of Link 1.

In Figure C.3, which shows the result of a change in the second flight size, we observe transients in both queues. Since the ACK rate of the first source is affected by the change in the queue size of Link 2 both queues are affected, even though Link 2 is downstream Link 1 from the point of view of the first source. We observe a good match between all three systems.

The queue response is qualitatively the same as for the two flow case in Section 5.2.2.

C.4 Case 2: cross traffic on Link 1

In this scenario UDP cross traffic is being sent over Link 1 utilizing half the capacity, i.e., $x_{c;1}(t) = c_1/2$. This time initially $f_1 = 1200$ and $f_2 = 1600$ packets. The plot in Figure C.4 displays the queue sizes when the congestion flight size of the first source is increased. The match between all systems are good. Note that the “lag” in the testbed data is due to the low sampling rate.

Figure C.5 shows the corresponding results when the second source’s flight size is perturbed. Ignoring the “lag”, we see that results are similar for all systems.

For this case as well, the queue response is qualitatively the same as for the two flow scenario in Section 5.2.2.

C.5 Case 3: cross traffic on Link 2

In this scenario UDP cross traffic is sending over Link 2 and utilizes half the capacity, i.e., $x_{c;2}(t) = c_2/2$, and initially $f_1 = 1500$ and $f_2 = 300$ packets.

The plot in Figure C.6 displays the queue sizes when the flight size of the first source is increased step wise. We observe a brief transient in both queues. This is different from the two flow case in Section 5.2.2 which had a step change in the first queue and where the second queue remained unaffected. The reason to the qualitatively different responses is due to the presence of the third flow with a flight size of 5 packets sharing Link 1 with Flow 1 to allow the delay to be measured. When Flow 1 increases its flight size, it slightly increases its sending rate on Link 1 at the expense of the measurement flow, causing the slight increase in the queue at the second link. This effect is not apparent in Figure C.2 because the measurement flow has a much smaller share of Link 1’s capacity, as seen by the higher link delay in that case.

The plot in Figure C.7 corresponds to the case when the second source’s flight size is increased by 200 packets. Here both queues are affected by the flight size perturbation, and the transient is significant for this case.

The agreement of the curves corresponding to the three systems is very good in the initial phase. There is however, a bias in the data as the system converges. The reason to this is likely the mismatch in the layer 4 capacities and cross traffic between the setups due to the unknown testbed packet overhead (we only know that is is approximately 10%), which implies slightly different system equilibriums.

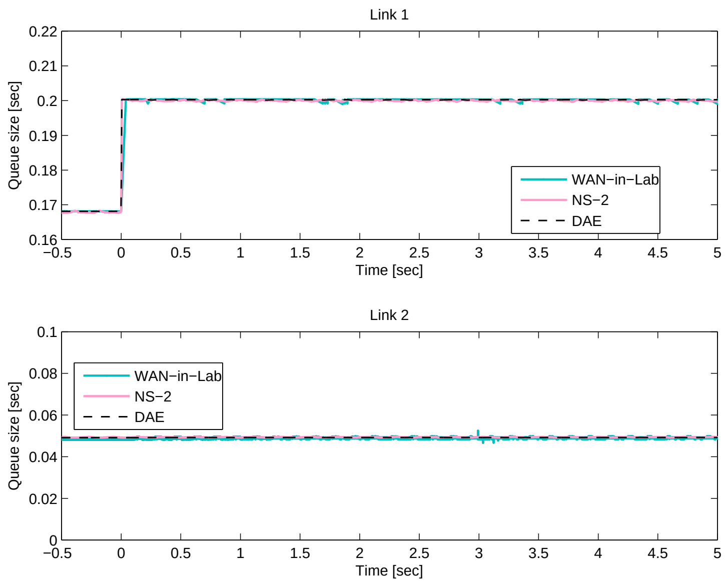


Figure C.2: Simulation: No cross traffic, step change in Window 1.

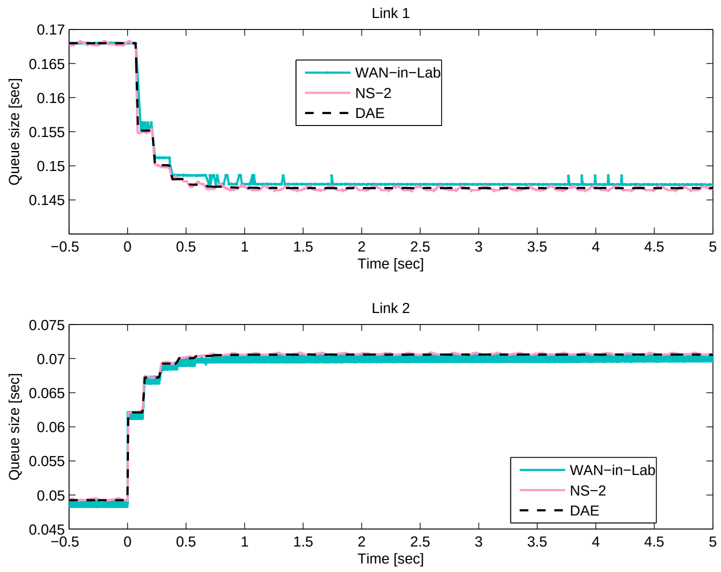


Figure C.3: Simulation: No cross traffic, step change in Window 2.

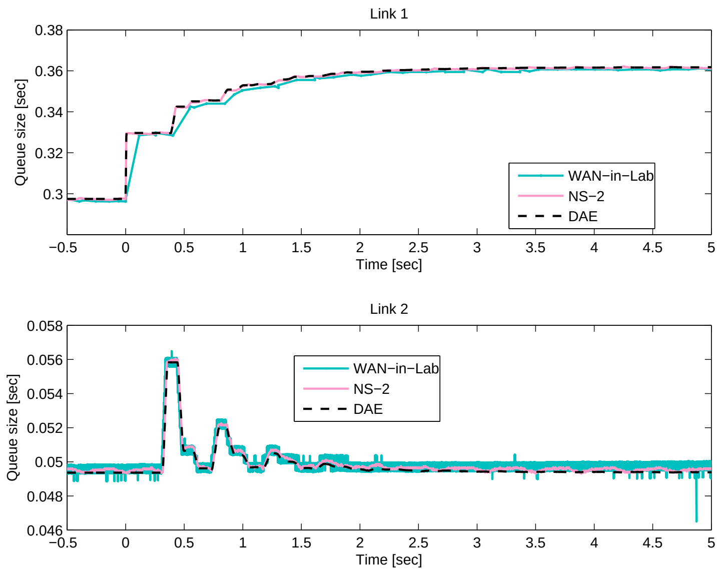


Figure C.4: Simulation: Cross traffic on Link 1, step change in Window 1.

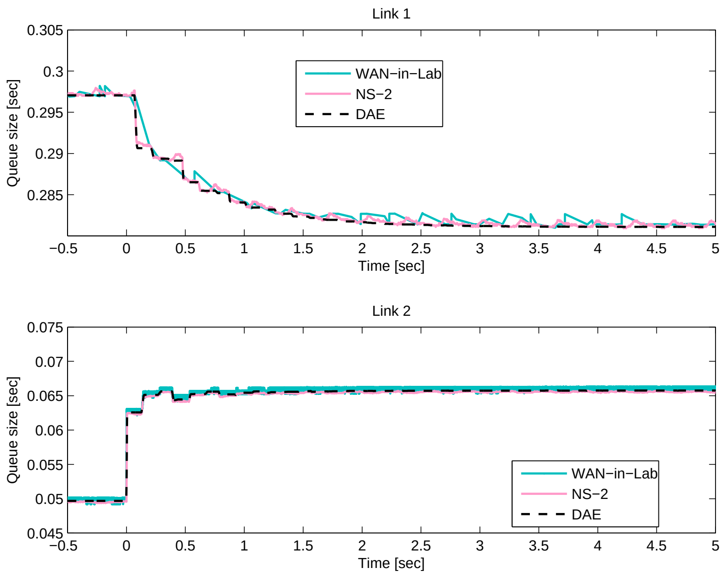


Figure C.5: Simulation: Cross traffic on Link 1, step change in Window 2.

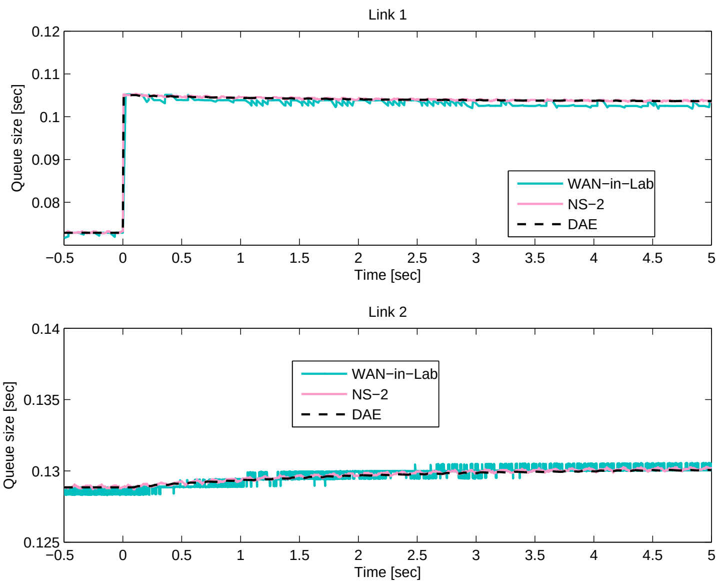


Figure C.6: Simulation: Cross traffic on Link 2, step change in Window 1.

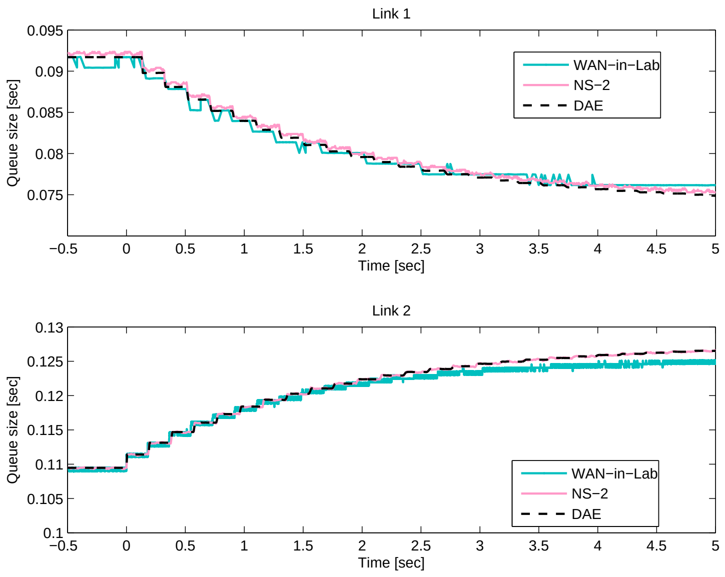


Figure C.7: Simulation: Cross traffic on Link 2, step change in Window 2.

C.6 Case 4: cross traffic on both links

In this scenario, UDP sources are sending over both links independently, each of them utilizing half the capacity of the link, i.e., $x_{c;1}(t) = c_1/2$ and $x_{c;2}(t) = c_2/2$. We also initially have $f_1 = 700$ and $f_2 = 600$ packets.

The plot in Figure 5.21 displays the queue sizes when the flight size of the first source is increased. Figure 5.22 shows what happens when a step is applied to the second source's flight size.

The testbed results agree with the predictions of NS-2 and the ACK-clocking model. Furthermore, the results are analogous to the case studied in Section 5.2.2.

Bibliography

P. Albertos and A. Sala Piqueras. 2002. *Iterative Identification and Control.* Springer-Verlag.

M. Allman and A. Falk. 1999. On the effective evaluation of TCP. *SIGCOMM Computer Communication Review*, 29(5):59–70.

M. Allman, V. Paxson, and W. Stevens. 1999. TCP congestion control. RFC 2581.

E. Altman, T. Başar, and R. Srikant. 1998. Robust rate control for ABR sources. In *Proceedings of IEEE Infocom 1998.*

E. Altman, C. Barakat, and V. Ramos. 2004. Analysis of AIMD protocols over paths with variable delay. In *Proceedings of IEEE Infocom 2004.*

K. J. Åström and B. Wittenmark. 1997. *Computer-controlled systems*, 3rd edition. Prentice Hall.

S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. 2001. REM: active queue management. *IEEE Network*, 15(3):48–53.

J. Aweya, M. Ouellette, and D. Y. Montuno. 2001. A control theoretic approach to active queue management. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 36(2-3):203–235.

F. Baccelli and D. Hong. 2000. TCP is max-plus linear and what it tells us on its throughput. *SIGCOMM Computer Communication Review*, 30(4):219–230.

F. Baccelli and D. Hong. 2002. AIMD, fairness and fractal scaling of TCP traffic. In *Proceedings of IEEE Infocom 2002.*

G. A. Baker and P. R. Graves-Morris. 1996. *Padé approximants.* Cambridge University Press.

D. P. Bertsekas, A. Nedić, and A. E. Ozdaglar. 2003. *Convex analysis and optimization.* Athena Scientific.

S. Bohacek, J. P. Hespanha, J. Lee, and K. Obraczka. 2003. A hybrid systems modeling framework for fast and accurate simulation of data communication networks. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems.*

R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. 2006. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494.

C. Boutremans and J.-Y. Le Boudec. 2000. A note on the fairness of TCP Vegas. In *Proceedings of International Zurich Seminar on Broadband Communications 2000.*

S. Boyd and L. Vandenberghe. 2004. *Convex optimization.* Cambridge university press.

B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. 1998. Recommendations on queue management and congestion avoidance in the Internet. RFC 2309.

L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. 1994. TCP Vegas: new techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications.*

C.-S. Chang, Y. Chiu, and W. T. Song. 2001. On the performance of multiplexing independent regulated inputs. In *SIGMETRICS '01: Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems.*

M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. 2007. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312.

H. Choe and S. H. Low. 2003. Stabilized Vegas. In *Proceedings of IEEE Infocom 2003.*

J. Y. Choi, K. Koo, J. S. Lee, and S. H. Low. 2005. Global stability of FAST TCP in single-link single-source network. In *Proceedings of the 44th IEEE Conference on Decision and Control.*

T. Cui and L. Andrew. FAST TCP simulator module for ns-2, version 1.1. `http://www.cubinlab.ee.mu.oz.au/ns2fasttcp`.

S. Deb and R. Srikant. 2003. Global stability of congestion controllers for the Internet. *IEEE Transactions on Automatic Control*, 48(6):1055–1060.

N. Dukkipati and N. McKeown. 2006. Why flow-completion time is the right metric for congestion control. *SIGCOMM Computer Communication Review*, 36(1):59–62.

F. Eng. 2007. *Non-Uniform Sampling in Statistical Signal Processing.* PhD thesis, Linköpings universitet.

W. Feng, D. D. Kandlur, D. Saha, and K. S. Shin. 1997. Techniques for eliminating packet loss in congested TCP/IP networks. Technical Report CSE-TR-349-97, U. Michigan.

W. Feng, D. D. Kandlur, D. Saha, and K. S. Shin. 1999. A self-configuring RED gateway. In *Proceedings of the IEEE Infocom 1999.*

W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha. 2002. The BLUE active queue management algorithms. *IEEE/ACM Transactions on Networking*, 10(4):513–528.

V. Firoiu, J.-Y. Le Boudec, D. Towsley, and Z.-L Zhang. 2002. Theories and models for Internet quality of service. *Proceedings of the IEEE*, 90(9):1565–1591.

S. Floyd. 1999. Validation experiences with the NS simulator. In *Proceedings of the DARPA/NIST Network Simulation Validation Workshop*, Fairfax, VA, May 1999. URL `http://www.icir.org/floyd/papers/NIST.Apr99.pdf`.

S. Floyd. 2000. Congestion control principles. RFC 2914.

S. Floyd. 2003. HighSpeed TCP for large congestion windows. RFC 3649.

S. Floyd, T. Henderson, and A. Gurtov. 2004. The NewReno modification to TCP's fast recovery algorithm. RFC 3782.

S. Floyd and V. Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413.

M. Fomenkov, K. Keys, D. Moore, and k claffy. 2004. Longitudinal study of Internet traffic in 1998-2003. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies.*

U. Forssell and L. Ljung. 1999. Closed-loop identification revisited. *Automatica*, 35 (7):1215–1241.

M. Gerla, M. Y. Sandidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo. 2001. TCP Westwood: Congestion window control using bandwidth estimation. In *Proceedings of IEEE Globecom 2001.*

H. Hjalmarsson. 2003. From experiments to closed loop control. In *Proceedings of IFAC Symposium on System Identification 2001.* Plenary address.

H. Hjalmarsson. 2005. From experiment design to closed loop control. *Automatica*, 41(3):393–438.

C. V. Hollot, V. Misra, D. Towsley, and W. Gong. 2002. Analysis and design of controllers for AQM routers supporting TCP flows. *IEEE Transactions on Automatic Control*, 47(6):945–959.

C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong. 2001a. A control theoretic analysis of RED. In *Proceedings of IEEE Infocom 2001*.

C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong. 2001b. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE Infocom 2001*.

C. Hunt. 1998. *TCP/IP: Network administration*. Cambridge: O'Reilly.

V. Jacobson. 1988. Congestion avoidance and control. *SIGCOMM Computer Communication Review*, 18(4):314–329.

K. Jacobsson, L. L. H. Andrew, A. K. Tang, S. H. Low, and H. Hjalmarsson. 2008. An improved link model for window flow control and its application to FAST TCP. *IEEE Transactions on Automatic Control*. To appear.

K. Jacobsson, H. Hjalmarsson, and N. Möller. 2006. ACK-clock dynamics in network congestion control – an inner feedback loop with implications on inelastic flow impact. In *Proceedings of the 45th IEEE Conference on Decision and Control*.

R. Jain. 1989. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *SIGCOMM Computer Communication Review*, 19(5):56–71.

H. Jiang and C. Dovrolis. 2005. Why is the Internet traffic bursty in short time scales? *SIGMETRICS Performance Evaluation Review*, 33(1):241–252.

C. Jin, D. X. Wei, and S. H. Low. 2004. FAST TCP: motivation, architecture, algorithms, performance. In *Proceedings of IEEE Infocom 2004*.

R. Johari and D. Tan. 2001. End-to-end congestion control for the Internet: delays and instability. *IEEE/ACM Transactions on Networking*, 6(9):818–832.

U. Jönsson, C.-Y. Kao, and H. Fujioka. 2007. A Popov criterion for networked systems. *Systems & Control Letters*, 56(9–10):603–610.

T. Kailath, A. H. Sayed, and B. Hassibi. 2000. *Linear estimation*. Prentice Hall.

D. Katabi, M. Handley, and C. Rohrs. 2002. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*.

kc claffy, S. O. Bradner, and S. D. Meinrath. 2007. The (un)economic Internet? *IEEE Internet Computing*, 11(3):53–58.

F. Kelly. 2003a. Fairness and stability of end-to-end congestion control. *European Journal of Control*, 9:159–176.

F. P. Kelly. 1999. Mathematical modelling of the Internet. In *Fourth International Congress on Industrial and Applied Mathematics, Edinburgh, Scotland.*

F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. 1998. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252.

T. Kelly. 2003b. Scalable TCP: improving performance in highspeed wide area networks. *SIGCOMM Computer Communication Review*, 33(2):83–91.

G. Kesidis and T. Konstantopoulos. 2000. Worst-case performance of a buffer with independent shaped arrival processes. *IEEE Communications Letters*, 4(1):26–28.

R. King, R. Baraniuk, and R. Riedi. 2005. TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP. In *Proceedings of the IEEE Infocom 2005.*

L. Kleinrock. 1975. *Queueing systems: Theory.* Wiley.

S. Kunniyur and R. Srikant. 2002. A time-scale decomposition approach to adaptive ECN marking. *IEEE Transactions on Automatic Control*, 47(6):884–894.

S. S. Kunniyur and R. Srikant. 2004. An adaptive virtual queue (AVQ) algorithm for active queue management. *IEEE/ACM Transactions on Networking*, 12(2): 286–299.

T. V. Lakshman and U. Madhow. 1997. The performance of TCP/IP for networks with high bandwidth delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350.

L. Le, J. Aikat, K. Jeffay, and F. D. Smith. 2007. The effects of active queue management and explicit congestion notification on web performance. *IEEE/ACM Transactions on Networking*, 15(6):1217–1230.

J.-Y. Le Boudec and P. Thiran. 2001. *Network Calculus.* Springer.

G. S. Lee, L. L. H. Andrew, A. Tang, and S. H. Low. 2007a. WAN-in-Lab: Motivation, deployment and experiments. In *Proceedings of PFLDnet 2007.*

J. Lee, S. Bohacek, J. P. Hespanha, and K. Obraczka. 2007b. Modeling communication networks with hybrid systems. *IEEE/ACM Transactions on Networking*, 15(3):630–643.

B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. S. Wolff. 1997. The past and future history of the Internet. *Communications of the ACM*, 40(2):102–108.

D. Leith and R. Shorten. 2004. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of PFLDnet 2004*.

I. C. Lestas and G. Vinnicombe. 2007. Scalable robust stability for nonsymmetric heterogeneous networks. *Automatica*, 43(4):714–723.

Y.-T. Li, D. Leith, and R. N. Shorten. 2007. Experimental evaluation of TCP protocols for high-speed networks. *IEEE/ACM Transactions on Networking*, 15 (5):1109–1122.

S. Liu, T. Başar, and R. Srikant. 2005. Pitfalls in the fluid modeling of RTT variations in window-based congestion control. In *Proceedings of IEEE Infocom 2005*.

S. Liu, T. Başar, and R. Srikant. 2006. TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks. In *Valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodologies and tools*.

L. Ljung. 1999. *System Identification: Theory for the User*, 2nd edition. Prentice Hall.

S. H. Low. 2000. A duality model of TCP and queue management algorithms. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*.

S. H. Low and D. E. Lapsley. 1999. Optimization flow control – I: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874.

S. H. Low, F. Paganini, and J. C. Doyle. 2002a. Internet congestion control. *Control Systems Magazine*, 22(1):28–43.

S. H. Low, F. Paganini, J. Wang, S. A. Adlakha, and J. C. Doyle. 2002b. Dynamics of TCP/RED and a scalable control. In *Proceedings of IEEE Infocom 2002*.

S. H. Low, F. Paganini, J. Wang, and J. C. Doyle. 2003. Linear stability of TCP/RED and a scalable control. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 43(5):633–647.

S. H. Low, L. Peterson, and L. Wang. 2002c. Understanding Vegas: a duality model. *Journal of ACM*, 49(2):207–235.

S. H. Low and R. Srikant. 2004. A mathematical framework for designing a low-loss, low-delay Internet. *Networks and Spatial Economics*, 4:75–101.

R. Ludwig and R. H. Katz. 2000. The Eifel algorithm: making TCP robust against spurious retransmissions. *SIGCOMM Computer Communication Review*, 30(1): 30–36.

L. Massoulié. 2002. Stability of distributed congestion control with heterogeneous feedback delays. *IEEE Transactions on Automatic Control*, 47(6):895–902.

L. Massoulié and J. Roberts. 2002. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Transactions on Networking*, 10(3):320–328.

M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. 1996. TCP selective acknowledgements options. RFC 2018.

M. Mathis, J. Semke, and J. Mahdavi. 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. *SIGCOMM Computer Communication Review*, 27(3):67–82.

F. Mazenc and S. I. Niculescu. 2003. Remarks on the stability of a class of TCP-like congestion control models. In *IEEE Proceedings of Conference on Decision and Control*.

V. Misra, W.-B. Gong, and D. Towsley. 2000. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *Proceedings of SIGCOMM '00*.

J. Mo, R. J. La, V. Anantharam, and J. C. Walrand. 1999. Analysis and comparison of TCP Reno and Vegas. In *Proceedings of IEEE Infocom 1999*.

J. Mo and J. Walrand. 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567.

N. Möller. 2008. *Window-based congestion control*. PhD thesis, The Royal Institute of Technology, KTH.

N. Möller, K. H. Johansson, and K. Jacobsson. 2006. Stability of window-based queue control with application to mobile terminal download. In *CD-ROM Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*.

J. Nagle. 1984. Congestion control in IP/TCP internetworks. RFC 896.

H. Newman, J. Bunn, D. Bourilkov, R. Cavanaugh, I. Legrand, S. Low, S. McKee, D. Nae, S. Ravot, C. Steenberg, X. Su, M. Thomas, F. van Lingen, and Y. Xia. 2006. The motivation, architecture and demonstration of the ultralight network testbed. In *Proceedings of CESNET 2006*.

ns2. The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

omnet. OMNeT++ Discrete Event Simulation System. `http://www.omnetpp.org`.

opnet. OPNET Modeler. `http://opnet.com/solutions/network_rd/modeler.html`.

T. J. Ott, T. V. Lakshman, and L. H. Wong. 1999. SRED: Stabilized RED. In *Proceedings of IEEE Infocom 1999*.

J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. 1998. Modeling TCP throughput: a simple model and its empirical validation. *SIGCOMM Computer Communication Review*, 28(4):303–314.

F. Paganini, Z. Wang, J. C. Doyle, and S. H. Low. 2005. Congestion control for high performance, stability, and fairness in general networks. *IEEE/ACM Transactions on Networking*, 13(1):43–56.

A. Papachristodoulou, L. Li, and J. C. Doyle. 2004. Methodological frameworks for large-scale network analysis and design. *SIGCOMM Computer Communication Review*, 34(3):7–20.

A. Pásztor and D. Veitch. 2002. PC based precision timing without GPS. *SIGMETRICS Performance Evaluation Review*, 30(1):1–10.

M. Peet and S. Lall. 2007. Global stability analysis of a nonlinear model of Internet congestion control with delay. *IEEE Transactions on Automatic Control*, 52(3): 553–559.

K. Ramakrishnan, S. Floyd, and D. Black. 2001. The addition of explicit congestion notification (ECN) to IP. RFC 3168.

RFC 793. 1981. Transmission control protocol. RFC 793.

I. Rhee and L. Xu. 2005. CUBIC: A new TCP-friendly high-speed TCP variant. In *Proceedings of PFLDnet 2005*.

L. Rizzo. 1997. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Computer Communication Review*, 27(1):31–41.

J. W. Roberts. 2001. Traffic theory and the Internet. *IEEE Communications Magazine*, 39(1):94–99.

W. J. Rugh. 1996. *Linear system theory*, 2nd edition. Prentice Hall.

S. Sahu, P. Nain, C. Diot, V. Firoiu, and D. Towsley. 2000. On achievable service differentiation with token bucket marking for TCP. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*.

J. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288.

R. Shorten, C. King, F. Wirth, and D. Leith. 2007. Modelling TCP congestion control dynamics in drop-tail environments. *Automatica*, 43(3):441–449.

S. Skogestad and I. Postlethwaite. 2005. *Multivariable Feedback Control*, 2nd edition. Wiley.

R. Srikant. 2004. *The Mathematics of Internet Congestion Control*. Birkhauser.

W. Stevens. 1997. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001.

M. Suchara, L. L. H. Andrew, R. Witt, K. Jacobsson, B. P. Wydrowski, and S. H. Low. 2008. Implementation of provably stable MaxNet. In *Broadnets 2008*. Submitted.

K. Tan, J. Song, Q. Zhang, and M. Sridharan. 2006a. A compund TCP approach for high-speed and long distance networks. In *Proceedings of the IEEE Infocom 2006*.

L. Tan, W. Zhang, G. Peng, and G. Chen. 2006b. Stability of TCP/RED systems in AQM routers. *IEEE Transactions on Automatic Control*, 51(8):1393–1398.

A. Tang, L. L. H. Andrew, K. Jacobsson, K. H. Johansson, S. H. Low, and H. Hjalmarsson. 2008. Window flow control: Macroscopic properties from microscopic factors. In *Proceedings of IEEE Infocom 2008*.

A. Tang, J. Wang, S. H. Low, and M. Chiang. 2007. Equilibrium of heterogeneous congestion control: existence and uniqueness. *IEEE/ACM Transactions on Networking*, 15(4):824–837.

P. M. J. Van den Hof and R. J. P. Schrama. 1995. Identification and control – closed loop issues. *Automatica*, 31(12):1751–1770.

G. Vinnicombe. 2002. On the stability of networks operating TCP-like congestion control. In *Proceedings of IFAC World Congress*.

J. Wang, A. Tang, and S. H. Low. 2004. Local stability of FAST TCP. In *Proceedings of the 43rd IEEE Conference on Decision and Control*.

J. Wang, D. X. Wei, and S. H. Low. 2005. Modelling and stability of FAST TCP. In *Proceedings of IEEE Infocom 2005*.

Z. Wang and J. Crowcroft. 1992. Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm. *SIGCOMM Computer Communication Review*, 22(2):9–16.

D. X. Wei, C. Jin, S. H. Low, and S. Hegde. 2006. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 14 (6):1246–1259.

X. Wei. 2007. *Microscopic behavior of TCP congestion control.* PhD thesis, California Institute of Technology.

J. T. Wen and M. Arcak. 2004. A unifying passivity framework for network flow control. *IEEE Transactions on Automatic Control*, 49(2):162–174.

L. Xu, K. Harfoush, and I. Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *Proceedings of IEEE Infocom 2004*.

I. Yeom and A. L. N. Reddy. 2001. Modeling TCP behavior in a differentiated services network. *IEEE/ACM Transactions on Networking*, 9(1):31–46.

L. Ying, G. E. Dullerud, and R. Srikant. 2006. Global stability of Internet congestion controllers with heterogeneous delays. *IEEE/ACM Transactions on Networking*, 14(3):579–591.

Y. Zhang, D. Leonard, and D. Loguinov. 2006. JetMax: Scalable max-min congestion control for high speed heterogeneous networks. In *Proceedings of IEEE Infocom 2006*.

K. Zhou, J. C. Doyle, and K. Glover. 1996. *Robust and optimal control.* Prentice Hall.

TRITA-EE 2008:020
ISSN 1653-5146
ISBN 978-91-7178-945-7