UNIVERSITE DE NICE-SOPHIA ANTIPOLIS - UFR SCIENCES

Ecole Doctorale STIC

THESE

Présentée pour obtenir le titre de

Docteur en SCIENCES

de l'Université de Nice Sophia Antipolis

Spécialité : Informatique

par

Chadi BARAKAT

Evaluation des performances du contrôle de congestion dans l'Internet

Soutenue publiquement le 4 Avril 2001 devant le jury composé de :

M.	Altman	Eitan	INRIA	Directeur
M.	Bernhard	Pierre	I3S	Examinateur
M.	Bonald	Thomas	France Telecom R&D	Examinateur
Mme.	Gravey	Annie	ENST-Bretagne	Rapporteur
M.	Lakshman	T.V.	Bell-Labs	Rapporteur
M.	Le Boudec	Jean-Yves	EPFL	Examinateur
M.	Nain	Philippe	INRIA	Examinateur

(14:00 - INRIA)

Table of contents

Chapter 1

Introduction

Since its creation in the early 70s, the Transmission Control Protocol (TCP) has the main objective to enhance the simple Best-Effort service provided by the IP (Internet Protocol) layer of the Internet architecture. Indeed, the IP layer implements a simple datagram delivery service without any guarantee in terms of the order of delivery, the correctness of the delivered packet, or the arrival of the packet to its destination. It is this simplicity of the IP layer that makes the success of the Internet and that permits an interconnection of a large number of transmission media (e.g., Local Area Networks, wireless networks, satellite networks, ATM (Asynchronous Transfer Mode) networks, etc.) into a single global network. TCP has been designed to build over this simple network layer, end-to-end channels that provide a reliable in-order data delivery service to the application layer [111]. It is an intelligent protocol located in hosts between the application and the simple IP layer, and it relies in its operation on the simple delivery service provided by IP and on the packets exchanged with the TCP peer in the receiving host. Figure 1.1 explains the role of the TCP protocol in the Internet architecture. Compared to the well known seven-layers OSI (Open Systems Interconnection) architecture of the ISO (International Standards Organization) [109], TCP corresponds to the transport layer known as layer 4 [1]. The TCP service is used by a large number of data transfer applications (e.g., FTP for file transfer, Telnet for remote login, SMTP for mail transfer, HTTP for web transfers, etc.) and is accessed via the socket interface of the operating system. This service avoids the implementation of the reliability mechanisms in every application which facilitates the task of network programmers.

The second objective of TCP upon its creation was to control the flow of packets so as not to overload the receiving host. This is called the end-to-end flow control of TCP [111]. At the epoch, the receivers were the main bottleneck in the network and it was necessary not to send more than the receiver buffer can hold. A window has been used for this purpose and its value is advertised by the receiver upon connection set-up [111]. At any moment, the sender is not

[1] In the Internet architecture shown in Figure 1.1, the application combines the three highest layers of the OSI model (application, presentation, session) into a single layer. The IP protocol corresponds to the network layer known as layer 3. The lowest layer of the Internet architecture, called Network in the Figure, corresponds to any transmission medium between two IP routers and combines the two lowest layers (data link, physical) of the OSI model.

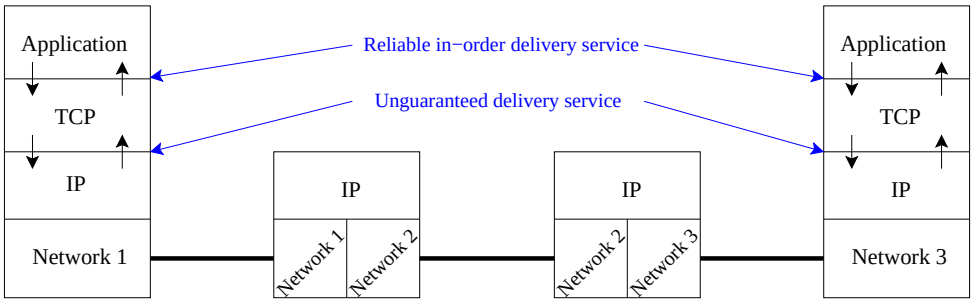


Figure 1.1: TCP in the Internet architecture

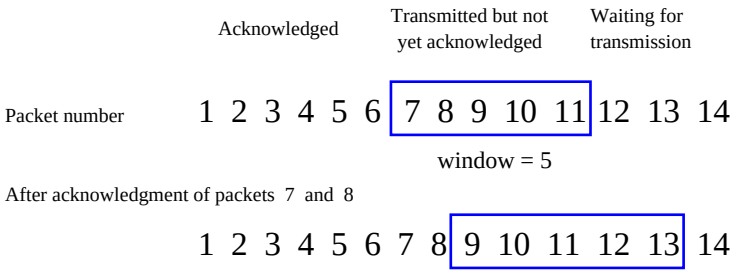


Figure 1.2: Window-based flow control

allowed to transmit more than the window size of data before the receipt of an acknowledgement from the receiver saying that it has delivered some data to the application. Once the sender receives an acknowledgement which says that the receiver has freed some place in its buffer, the window slides and the sender can transmit more data. Such way to control the flow of packets is called window-based flow control compared to a rate-based flow control where the source transmits data continuously at an appropriate rate (e.g., ABR (Available Bit Rate) service in ATM networks). Figure 1.2 depicts an example of a window-based flow control protocol. The window is represented by the rectangle which indicates the numbers of packets the source can transmit before the receipt of the next acknowledgement. The left-hand side of the window indicates the lastly acknowledged packet which slides to the right when a packet is acknowledged allowing the right-hand side of the window to cover new packets (packets 12 and 13 in the figure). Indeed, the right-hand side of the window is obtained by adding the window size to the left-hand side. Once the right-hand side of the window covers a new packet, the source is allowed to inject it into the network.

This version of TCP has served the Internet for years. Applications using TCP got a reliable connection-oriented in-order service at a rate determined by sender and receiver capacities. The network status was not considered at all by TCP sources; a TCP source injected new data into the Internet whenever it finds an empty place in the receiver buffer. This worked well since the amount of Internet traffic was small compared to the bandwidth available on network links.

But, the Internet grew after that (and is continuing at a fast rate) and the traffic exceeded the network capacity. Routers became unable to forward packets at the rate they arrive to their interfaces. This has resulted in a series of congestion collapses by the middle of the 80s; Internet routers dropped most of the packets and TCP sources insisted on retransmitting the lost packets as well as new packets without backing off their transmission rates. This has motivated Van Jacobson in 1988 [75] to introduce his well known algorithms which delegate to TCP another important task in addition to its previous tasks, that is the control of the rate of packets as a function of network conditions. This has been done again with a window-based approach. A variable window has been added to TCP [75], called *congestion window*, that increases when the network is not congested and that backs off when a congestion occurs. Another mechanism has also been added [75] to back off the retransmission timer[2] when the network is severely congested. The congestion of the network is inferred at the source from the loss of packets. This congestion detection method came in compliance with the philosophy of the Internet architecture; the network must be as simple as possible and it is for hosts to get the information they require for their operation from the end-to-end communication between them. The congestion window is always bounded by the window advertised by the receiver so that TCP continues to do its end-to-end flow control. In the following chapter, we will present with some details the different mechanisms proposed in [75] for congestion control in the Internet.

The addition of the congestion control algorithms to TCP has made of this protocol the main part of the Internet and the main warranty for its stability and its survival. A strong relation was born between TCP congestion control and the Internet so that some people call the Internet architecture the TCP/IP architecture and other people confuse between TCP and its congestion control algorithms. The importance of TCP can be seen from the importance of its contribution to total Internet traffic; except a small number of packets sent over the Internet for administration, test, and control purposes, most of Internet traffic is of TCP type (95% of all bytes and 85 - 95% of all packets [123]). It has been also recommended [62] that applications implementing their own congestion control should do it in a TCP-friendly manner. This strong relation between TCP and the fate of the Internet has opened the way for a huge amount of research works on the performance of the TCP protocol. Obviously, most of these works have been devoted to the congestion control part of the protocol. Many researchers have studied during the last decade how well TCP behaves and what are the required mechanisms to add to TCP or to the network in order to approach the optimal behavior. An optimal congestion control in a best-effort network as the Internet should lead to:

- An efficient utilization of network resources (i.e., available bandwidth).
- A low level of congestion in network routers (i.e., small queues and low drop probabilities).

[2] As we will see later, a TCP source schedules a retransmission timer when a packet is transmitted to infer the loss of the packet when the timer expires and the acknowledgement of the packet is not yet received.

- A fair sharing of resources between the different flows.

Note that different fairness schemes could be envisaged for sharing network resources, see [89] for a discussion of this issue. In the next chapter, we will address with some details the fairness of TCP congestion control.

The different works on TCP have been motivated by the degradation in performance the protocol started to experience with the growth of the Internet. TCP is required to achieve the above objectives whatever is the size of the Internet and whatever are the transmission media crossed by Internet traffic. At the beginning of the last decade, TCP was performing well since the Internet was not so complicated and the traffic was not so important. Low-speed good-quality terrestrial lines (e.g., the 56 Kbps lines of the ARPANET backbone) were connecting some routers in the United States. The last years however we have seen a tremendous growth of the Internet which is now covering most of the planet and involving most of the markets. Moreover, different transmission media have been introduced into the Internet which has considerably increased the heterogeneity of the network. This has given to some Internet paths completely different characteristics than those of the simple terrestrial network TCP congestion control has been designed to. As an example of these new transmission media we cite: high speed links (fiber optics), low speed links (dial-up modem lines), long and variable delay paths (satellite links), lossy links (wireless networks), asymmetric paths (hybrid satellite networks), etc. This explosion of the Internet in terms of size, traffic, and heterogeneity has complicated the task of TCP which is now required to ensure the stability of a large environment and at the same time to provide a good service to all Internet users. The different research works that have been done in the last decade have revealed certain problems with the protocol and they have proposed different solutions. Some solutions only propose modifications to the TCP protocol while others propose to help the protocol with some mechanisms inside the network. In the latter case, we can distinguish between solutions that propose changes to the network without any change to the TCP protocol — some authors tolerate changes to receivers (i.e., user machines) but insist on not changing the sources (i.e., Internet servers) — and solutions that propose at the same time changes to the network and to the TCP code. Some of the solutions (e.g., modifications to the error recovery phase) have found their way into standardization[3] while others are in the validation process. One should expect that other problems and other solutions are going to be found given the fast growth of the Internet and the continuous increase in its heterogeneity.

By looking at the different works on TCP performance, we can easily distinguish two main approaches:

- The end-to-end approach which supposes that the network between the two hosts is a black box whose content we don't know. TCP mechanisms are evaluated and improved as a function of the different forms under which packets leave the box. Some of the works in

[3]The Internet Engineering Task Force (IETF) is the standardization organization of the Internet and standards are published as RFCs (Request For Comments), http://www.ietf.org/.

this direction are compliant with the original congestion control mechanisms and came as a correction of some mistakes. Other works have been motivated by the heterogeneity of the Internet and the idea that TCP must cope with the new transmission media without any hint from the network. These latter works propose modifications to TCP congestion control itself. As examples of works following this approach we find those that aim to improve the error recovery phase of the protocol (e.g., [64, 90]), those that aim to improve the slow start phase (e.g., [2, 19, 72]), etc.

- The network-specific approach which consists in working with some particular network type (e.g., satellite network, wireless network, a particular buffer management strategy) and improving TCP performance by adding some mechanisms to the network. The TCP protocol in the hosts may contribute to the solution via a specific feedback sent from the network that specifies the action it must take. Typical examples of works in this direction are those that split the TCP connection within the network (e.g., [26, 24, 70]), those that add explicit congestion and loss notifications to TCP sources (e.g., [26, 53, 61]), etc.

The different works on TCP can be further classified into four categories as a function of the network characteristic they deal with:

- Works that study the performance of TCP in a general data network without looking at a particular transmission technology. We find here all the works that correct problems in the original mechanisms of TCP as well as the works that try to improve the congestion control in the Internet by changing the TCP itself or by adding some mechanisms to routers (e.g., by improving the strategy with which Internet routers manage their buffers at the moment of congestion). The growth of the Internet is the main motivation behind these works.

- Works that study the impact of the round-trip time on TCP performance. The focus is on the increase in the round-trip time on some paths or on the difference in round-trip time between concurrent TCP connections. Typical examples of such works are those that deal with satellite networks.

- Works that study the impact of bad quality links (e.g., noisy links) on TCP performance. Typical examples are the works on TCP over wireless networks.

- Works that study the impact of the slowness of the return path of a TCP connection. Works on hybrid satellite networks and cable networks are typical examples of such category.

We adopt this classification in four categories when we present later an overview of the different works on TCP performance. With this abstraction of the network by the characteristics of the path of the connection, we are able to place together the different contributions that treat the same problem but that are presented in the literature in different environments. For example,

different works have studied the impact on TCP of the long round-trip time in satellite networks. These works will be also useful for TCP connections crossing long terrestrial lines.

Concerning the tools that people use in the study of TCP performance, we can distinguish three main ones:

- *Experimentations*: Real TCP connections are run over the Internet or some experimental network. In the latter case, a background traffic needs to be generated to emulate the exogenous traffic in the Internet.

- *Simulations*: All the network including the TCP protocol is emulated in software. The widely used simulator for TCP studies is the `ns` simulator [102]. This is an event-driven network simulator developed at Lawrence Berkeley National Laboratory and which implements with a good precision the code of TCP and other Internet protocols.

- *Modeling*: Analytical models are associated to TCP and to the network. Generally, stochastic processes are used to model parts of the Internet of which we cannot predict exactly the behavior (e.g., the exogenous traffic, the round-trip time, the moments at which a congestion is detected at a TCP source). Deterministic models are however used for TCP which does not contain any randomness. These analytical models are solved for some performance measures, for example for the average transmission rate, and some conclusions are made on the factors impacting the performance of TCP.

Given the importance of the TCP protocol, we address in this thesis the general problem of TCP performance evaluation. The congestion control part of TCP is considered. We focus mainly on the modeling tool, but we use the simulation and experimentation tools for validation of our analytical results. We shall consider the two approaches for TCP study. First, we look at the performance of TCP from an end-to-end point of view. We elaborate some sophisticated models where we account for the different parameters that may impact the performance of TCP. Using the machinery of stochastic processes, we solve these models for simple explicit expressions of TCP average transmission rate. The average transmission rate, also called the *throughput*, is the main performance measure that indicates how well a bulk TCP transfer (e.g., a file transfer) is done. In the following chapters, we explain the utility of finding explicit expressions for TCP throughput. Our work in this direction can be seen as a general framework for TCP modeling that explains the different issues to be considered and that is in compliance with the previous works in this domain. Our different results are validated via measurements we made on real TCP connections over the Internet.

In the second part of this thesis, we get inside the network and we look at the performance of TCP in some particular media. We choose the three network types that are considered by researchers as the most challenging for TCP: large bandwidth-delay product networks, asymmetric networks, and wireless networks. In the first network type, we focus on the case when the

round-trip time is large and when the buffering capacity in network routers does not scale with the bandwidth-delay product. For each environment, we explain the problem of TCP and we try to optimize the solution considered by researchers as the most appropriate for TCP performance enhancement. We develop some analytical models for this purpose that capture the characteristics of the different networks. Our models show that the proposed solutions are not optimal and that the performance can be further improved by some additional mechanisms. We explain how the existing solutions need to be tuned for a better performance and we validate our conclusions via simulations. Then, we propose for the large bandwidth-delay product network and for the asymmetric network, two solutions that can be easily implemented and that solve the problems with the existing solutions. As in our work on end-to-end modeling, we try in this part to stay as general as possible and to give a framework for how to optimize the performance of TCP. Our contribution in this direction figures in the problems we will identify, in the analytical models we will develop, in the guidelines we will provide, and in the practical solutions we will propose.

In the following chapter and based on our classification of the different works on TCP performance, we present an overview of what has been done in this domain since the introduction of the congestion control mechanisms in 1988 [34]. In the third chapter, we address the problem of end-to-end modeling of TCP. We first explain the usefulness of this approach, then we discuss the different issues to be considered when modeling TCP [27]. After that, we present our model for TCP dynamics which is based on a fluid approach. We also present two approaches for modeling the network between the TCP peers: a simple approach based on some Markovian assumptions and a general approach. In Chapters 4, 5 and 6, we solve these models for the explicit expression of the throughput and we show [11, 12, 13, 15, 16, 27] via measurements over the Internet the gain we obtain from considering a sophisticated model for the network rather than a simple model as has been done in the literature. The validation of our results is based on the technique for validation we introduced earlier. In the sixth, seventh, and eighth chapters we pass, to our network-specific study of TCP. Respectively, we present our contributions in case of large bandwidth-delay product networks [29], asymmetric networks [31], and wireless networks [32]. These chapters contain a quick overview of TCP problems in the three environments as well as an explanation of the particular problems we will address. They also contain our analytical models, the practical solutions we came with, and the simulation results. We end this thesis with a chapter containing our conclusions and some perspectives on our future work. Partial conclusions and perspectives are also provided at the end of each chapter of this thesis.

Chapter 2

TCP congestion control evolution

We summarize in this chapter the mechanisms of TCP as well as the different works on TCP congestion control since its introduction in 1988. Some of the works on TCP consider a general network and don't limit themselves to a particular transmission technology. Typical examples of such works are those aiming to improve the loss recovery capacity of TCP [4, 56, 64, 90]. The other works deal with some particular environments as the satellite environment [3, 7] or the wireless environment [26]. Given that a contribution in one environment can be useful in another environment possessing the same characteristics, we chose to make the summary of the works on TCP independently of the transmission media carrying Internet traffic. We group the different contributions as a function of the network characteristic they deal with. We identify [34] for this reason four characteristics of an Internet path that we consider as the most challenging for TCP and as the real motivation behind most of the works in the literature. These characteristics are: the bandwidth-delay product, the round-trip time, the rate of losses not caused by congestion, and the degree of bandwidth asymmetry between the forward and the reverse directions of the TCP connection path. By bandwidth-delay product of a path we mean the product of the *bottleneck bandwidth* on this path[4] times the two-way propagation delay. We don't include in this product the number of packets that can be queued in network routers. When studying this product, we will focus on the bandwidth component and study how TCP can profit from the fast increase in transmission rates. By the round-trip time characteristic, we mean paths that have long or different round-trip times. The rate of losses not caused by congestion refers mainly to paths containing some bad quality links where packets can be lost due to transmission error or link outage. When the congestion control has been added to TCP for the first time, these latter paths were supposed not to exist in the Internet. Losses were only supposed to be caused by buffer overflow in network routers [75]. In [84], losses caused by a transient congestion of a router are also considered as non-congestion losses. A transient congestion means a non-loaded router that drops some packets due to their arrivals in bursts. The fourth characteristic we consider refers to asymmetric networks where the return path from the TCP receiver to the TCP sender

[4] Paxon in [107] defines the bottleneck bandwidth as the upper bound for how fast any connection can transmit along the path due to the data rate of the slowest forwarding element along the path.

is not fast enough to carry the flow of informations required for TCP operation.

In what follows we summarize the different works on TCP based on the previous classification. For every one of the four characteristics, we explain its impact on TCP performance and the different solutions that have been proposed. We also mention some works on TCP that are not really related to these characteristics. But before doing that, we outline the basic mechanisms of TCP [111] as well as the congestion control algorithms added to TCP in 1988 [75]. The material in this chapter has been published in [34].

2.1 Overview of TCP mechanisms

Before the addition of congestion control to TCP, the main objectives of the protocol were (and still are) the end-to-end flow control and the error control. The addition of congestion control has added a third important objective to TCP. We will explain in the following sections how these three objectives are achieved by the protocol.

2.1.1 End-to-end flow control

The end-to-end flow control means that the sender must not inject into the network more than the receiver can hold in its buffer. This will prevent a situation where a packet arrives at the receiver and does not find a place to be queued in order to be held later to the application. A sender must account for the worst case where all packets transmitted but not yet acknowledged arrive at the destination and find that they have to wait a little because the application is not ready to read them. As we explained in the previous chapter (Figure 1.2), this control is done with a window that limits the number of packets the sender can inject into the network and that slides when the sender receives an acknowledgment saying that the receiver has correctly received some data and handed them to the application. This window is advertised by the receiver at connection set-up and it is updated during the connection lifetime if the buffer space at the receiver changes. Figure 2.1 shows the field in the TCP header (`window`) that is used to carry this information. It is a 16-bit field telling the sender the size in bytes of the window it must use. Note here that the two sides of the window (Figure 1.2) can slide independently of each other. Indeed, the left-hand side of the window slides to the right when data arrive correctly and in-order at the receiver buffer. The right-hand side, obtained by adding the window field to the left-hand side, slides to the right when data are handed to the application. It is the right-hand side which indicates whether the TCP source can transmit new packets or no.

2.1.2 Error control

The error (or loss) control means that TCP is responsible for the recovery from any loss of information inside the network (or even in the hosts below the TCP layer). This is done by retransmitting the lost information until it reaches correctly the receiver. The detection of

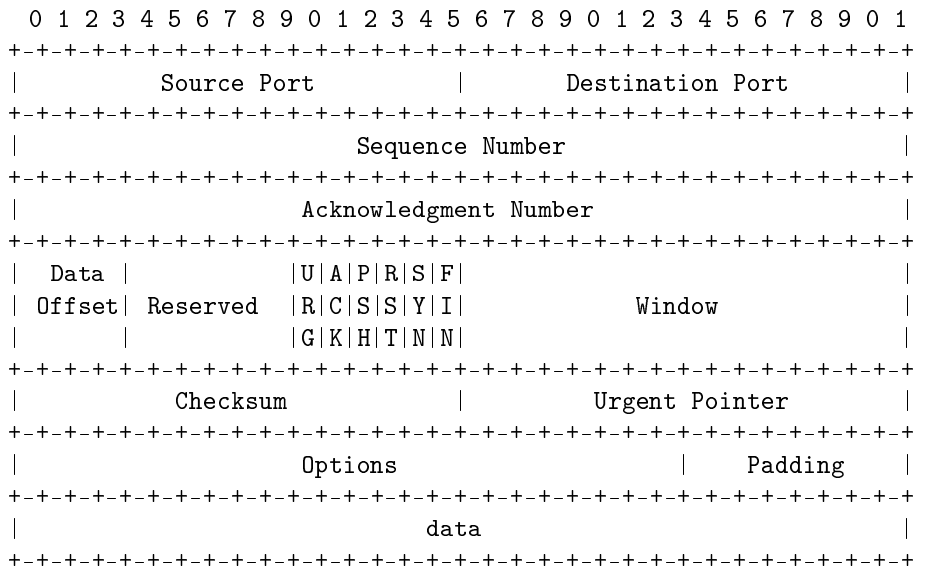


Figure 2.1: TCP header format (one tick mark represents one bit position)

the loss in TCP is based on sequence numbers, a retransmission timer and cumulative positive acknowledgments (ACK). The TCP receiver acknowledges the receipt of data packets. Usually, one data packet over two is acknowledged [41]. This is called the *delay* mechanism and its objective is to reduce the volume of ACKs generated by a TCP receiver. An ACK carries (implicitly) the sequence number of the last in-order byte received. This forms a cumulative information robust against ACK loss. An ACK sent in response to an out of order packet is called a *duplicate* ACK since it carries the same information as the previous ACK. A TCP receiver is asked to transmit a duplicate ACK in response to every out of order packet [41]. With these ACKs, the TCP source tries to estimate the round-trip time of the connection and sets accordingly a timer when packets are sent. An Exponentially Weighted Moving Average algorithm is used to smooth the oscillations in the round-trip time [111]. The expiration of the timer, called a *timeout*, before the receipt of an ACK is considered by the sender as a loss signal. The first packet sent and not yet acknowledged is directly retransmitted. Another mechanism has been added to TCP to detect losses using duplicate ACKs without the need to wait for a long timeout. It is called *Fast Retransmit* [75, 121]. According to this mechanism, the receipt of four consecutive ACKs carrying the same sequence number — the original plus three duplicate ACKs — indicates the loss of the packet following the lastly acknowledged packet. The receipt of a duplicate ACK means that a packet transmitted after the packet the source is waiting for its acknowledgement, has reached the destination which forms an indication of a potential loss. The number four is chosen in order to minimize the probability that a reordering of packets — which may exist in the Internet [37] — causes a wrong error detection.

2.1.3 Congestion control

Congestion control in data networks means that the different sources must adapt their transmission rates as a function of network load. The control must be done in a distributed manner and must achieve the three objectives we mentioned in the previous chapter: to well utilize the network resources while not bringing the network into congestion and while sharing fairly the available resources between the different flows [50, 78]. We can add here another requirement for a congestion control mechanism, that is the fast convergence to a fair regime when a new flow arrives or an existing flow vanishes.

There is quite a consensus in the research community that additive-increase multiplicative-decrease policies are appropriate for a distributed congestion control [50, 75]. The different flows have to increase their rates in an additive manner (i.e., linearly) in the absence of congestion and to decrease them with a multiplicative factor at the moment of congestion. A condition for the convergence to a fair regime is that all the flows must increase their rates with the same slope and decrease them with the same factor. Such condition has been established in [50] for flows that reduce their rates at the same moments and that have to share equally the available bandwidth. TCP is supposed to implement such a control policy. Other congestion control protocols as the ABR (Available Bit Rate) mechanism in ATM networks [21] use similar policies.

The rate control in TCP is done by changing the size of a congestion window that has been added to the protocol [75] and that also limits the number of packets the source can transmit before the receipt of an acknowledgement. Of course, this window is upper bounded by the receiver window for end-to-end flow control purposes [121]. By changing the size of the congestion window, the source changes its transmission rate given that the rate on a TCP connection (or of any other window-based flow control mechanism) can be approximated at any moment by the window size (which equals the number of packets the source has in the network) divided by the round-trip time [18, 84].

Two algorithms are used by TCP for window increase [75, 121]: the *congestion avoidance* algorithm which is the additive-increase component of the protocol, and the *slow start* algorithm which is a transitory phase that aims to bring the connection safely and quickly into the congestion avoidance mode. Congestion avoidance is the mode in which the TCP connection is supposed to stay once it gets out of the slow start phase. For congestion detection and window reduction, it has been chosen [75] to divide the window by two in both modes when a packet is lost. Packets are supposed to be only lost (or at least in most of the cases) due to buffer overflow in network routers. Thus, the loss of packets forms an implicit congestion signal that the TCP sources use for their control. The mapping between packet losses and congestion signals is another issue that many works have addressed. As we will explain later, these works are based on the idea that all losses in a window of packets (i.e., packets of the connection which are present at the same time inside the network) should be considered by the TCP source as a single congestion signal.

This method for congestion detection has been chosen for its simplicity; no additional mechanisms have to be added to the network which facilitates the deployment of TCP congestion control[5]. The network is still considered as a black box whose content we don't know and which we don't ask for any explicit feedback. However, it has been shown that the absence of explicit feedback from the network, combined with the simple drop-tail policy[6] in network routers, is the source for many problems. We will cite here three of them and try to present in the following sections the solutions that have been proposed to each problem.

- The first problem is that the TCP sources continue to increase their rates until the network becomes congested and here they react very conservatively by dividing their windows by two. This results in the network oscillating periodically between periods of congestion where the routers drop many packets and periods of non-congestion where packets are not dropped. Long queues build up during periods of congestion and the network becomes under-utilized during periods of non-congestion (except if there are enough packets queued in routers upon congestion so that the network remains well utilized after the reduction of the load). It would be better instead to maintain the load uniform so that the utilization of network resources are maintained at high values and the queues in network routers at small sizes. This is the main objective of active queue management techniques proposed to replace the simple drop-tail ones [42, 65]. These techniques form a network-level solution for one of the problems of TCP congestion control.

- The second problem with the congestion detection method of TCP is that all losses have to be caused by buffer overflow in order to get good performance. This is the main reason for the deterioration of TCP performance in noisy environments as wireless networks [26].

- The third problem with this method, which is the normal cost to pay for the absence of an explicit congestion notification and a sophisticated mechanism at the TCP source that anticipates the congestion of the network, is the need to retransmit all packets that are dropped in network routers. It would be better to anticipate the congestion in network routers and to send explicit congestion signals to TCP sources so that these retransmissions could be avoided. Some works in the literature [45, 78, 127] propose instead to anticipate the congestion at the sources without the need for an explicit feedback from the network. This has been simply done by observing the variation of the round-trip time as a function of the transmission rate (or the window size).

Let us now discuss with some details the different mechanisms of TCP congestion control [75, 121]. We look for this purpose at the different possible scenarios of the evolution of the congestion window since the beginning of the connection. These scenarios are summarized in Figure 2.2

[5]The incremental deployment is one of the main requirements for a new mechanism to be accepted in the Internet community.

[6]A drop-tail buffer drops an incoming packet when there is no place in the buffer to queue it.

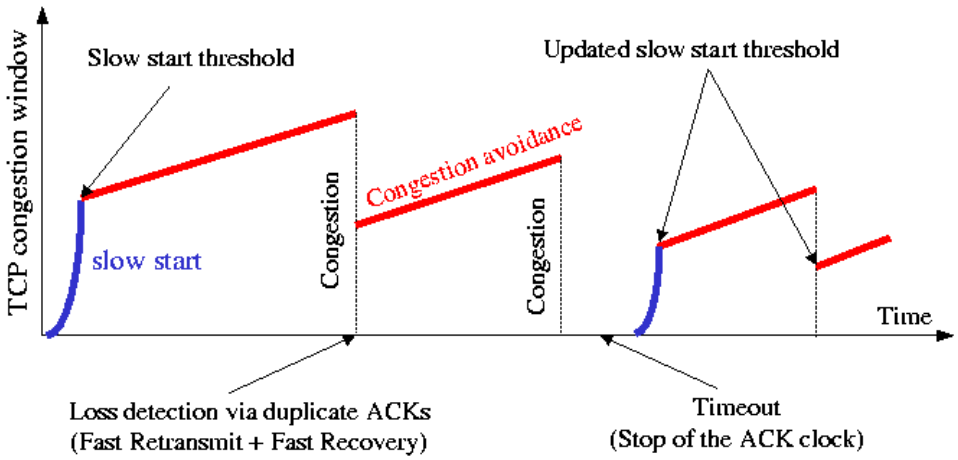


Figure 2.2: Mechanisms of TCP congestion control

for a long life TCP connection. We assume that the receiver window is large so that it does not limit the rate of the connection. A TCP source always holds an estimate of the network capacity that it tries to update during the lifetime of the connection. By capacity of the network we mean the maximum number of packets (or of bytes) that can be fit on the path between the source and the destination. This is called in the literature the *pipe size* [84]. It is equal to the bandwidth-delay product plus the maximum number of packets that can be queued in network routers. At the beginning of the connection the source sets its estimate to a default value. Most of TCP implementations make the assumption that the receiver is the only bottleneck and set their estimate to the window the receiver advertises [10]. Starting from one segment[7], or a larger value as we will see later, the source calls the slow start algorithm. During slow start, the congestion window is increased by one segment for every new ACK (i.e., non duplicate ACK) until the source estimate of the network capacity is reached. This results in an exponential increase of the congestion window; if all packets are acknowledged and if all ACKs arrive at the source, the window will double every round-trip time. The network capacity estimate is called *the slow start threshold* since it defines the end of slow start. We are in the slow start mode whenever the congestion window is less than the slow start threshold [75]. In the sequel we denote this threshold by W_{th}[8].

Although the window increase is fast during slow start, the increase is still slow compared to a direct transmission at the beginning of the connection at a window equal to W_{th}. It is from here that came the name slow start [75] since the start of the transmission before the addition of congestion control were directly at the window advertised by the receiver. Given that TCP transmits all packets allowed by its window in a burst without any spacing, starting

[7] A segment is the size of data in a TCP packet.

[8] Most often the congestion window is denoted by `cwnd`, the receiver window by `rwnd`, and the slow start threshold by `ssthresh`.

directly at W_{th} would result in a long burst of packets and a heavy overload on the network. Packets from the connection as well as from other connections could be lost which deteriorates the overall performance. Slow start aims to alleviate this burstiness while trying to fill as quickly as possible the network capacity. Moreover, the capacity of the network may be overestimated. A loss then occurs during slow start before reaching W_{th}. Here, the source updates W_{th} and sets it to half the current window size. Slow start serves then in this case as a means to get a better estimate of the network capacity. This estimation role of slow start is more pronounced at the beginning of the connection where no idea on the network is available and where W_{th} might be set to more than what the network can support. But, given the relatively fast window increase during slow start, this estimation procedure has been shown to impact negatively the performance of the connection as well as that of the other connections [72]. The TCP source has to bring the network into severe congestion in order to gauge its capacity. Many packets are lost due to the exponential window increase and the source spends long time retransmitting the lost packets before resuming the transmission at the new capacity estimate. It has been proposed instead [72] to estimate the capacity of the network at the beginning of the connection via additional mechanisms (e.g., by using the flow of ACKs) and to set the slow start threshold to a more appropriate value. This should avoid the congestion during slow start and improve the performance. In the chapter on TCP performance in large bandwidth-delay product networks, we will address this issue in more details [29]. Note here that all the studies on the slow start phase hold for short transfers for which the slow start phase forms an important part. Long transfers, often called long-life connections [92, 96, 97, 105], are mostly impacted by the additive-increase multiplicative-decrease behavior of the protocol and hence the slow start phase can be ignored.

Once W_{th} is reached, the source switches to the congestion avoidance mode where the congestion window is increased slowly by one segment for every window's worth of acknowledgments [75, 121][9]. This rate has been chosen [75] to get a constant increase in the congestion window every round-trip time; if all packets are acknowledged and if all ACKs arrive at the source, the congestion window will increase by approximately one segment every round-trip time. If the round-trip time does not change, the rate of the TCP flow during congestion avoidance will vary in time in an additive-increase multiplicative-decrease manner. However, this will not be the case when the round-trip time starts to grow with the congestion window [27]. On paths where the TCP connection has an important share of the available bandwidth, the increase in the congestion window results in an increase in network load and round-trip time which makes the rate increase sub-linear instead of linear [12, 18, 27, 78, 84]. The performance will then be different than what we would obtain if the rate increase were linear. We will address this problem later when we study the end-to-end modeling of TCP. Note here that this change of the rate increase from linear to sub-linear is the result of the fact that TCP congestion control is based on changing

[9] Let W denote the window size in segments. During congestion avoidance, W is increased by $1/W$ when a non-duplicate ACK is received. This results in an increase in W by approximately one segment when W non-duplicate ACKs are received.

the window size rather than changing the rate. The change in rate is a function of the round-trip time of the connection. This dependency of the rate increase on the round-trip time results in the famous problem of the unfairness of the protocol in presence of connections of different round-trip times [60, 84]. The window variation of TCP can be considered as additive-increase multiplicative-decrease in the round-trip time space but not in the time space.

In contrast to slow start which aims to fill as quickly as possible the network capacity, congestion avoidance aims to probe slowly the network for any extra bandwidth [75]. The probe continues until a loss occurs. Here, the source supposes that the network is congested and sets its estimate of the capacity to half the current size o the window. The first version of TCP that implements congestion control, called Tahoe [56, 75], sets the congestion window at this point to one segment and uses again slow start to reach the new estimate of the capacity. But, slow starting after every congestion event deteriorates the performance given the small window, and hence the low bandwidth utilization, during the slow start phase. This slow start is necessary when the loss is detected via timeout; a timeout means that the ACK stream has stopped (otherwise the loss would be detected with Fast Retransmit [121]) and slow start must be used to fill again smoothly the network capacity. However, in the Fast Retransmit case, slow start could be avoided since the ACK stream is still active. The source can inject new packets into the network upon ACK arrivals so that the network does not drain and the utilization is kept at high values. New packets are injected at a rate controlled by the new value of the congestion window after the reduction. They are also injected in a network-friendly manner given the smooth rate at which ACKs arrive at the source (the rate of ACKs matches the available bandwidth in the network). The avoidance of slow start in case of Fast Retransmit is exactly what the later versions of TCP (Reno [121], New Reno [64], SACK [56], FACK [90], etc.) try to do. When three duplicate ACKs are received, these versions call a *Fast Recovery* algorithm. Fast Recovery tries to retransmit the losses in the current window while maintaining a number of packets in the network equal to the new estimate of network capacity. Once losses are recovered, this algorithm ends and a normal congestion avoidance phase is called without the need for slow start. As it is always the case with TCP, new packets and retransmissions are transmitted during Fast Recovery only upon the receipt of ACKs. If Fast Recovery fails, the ACK stream stops, a timeout occurs and the source resorts to a slow start phase as in the Tahoe version. The variety of versions of TCP is due to a variety of implementations of the Fast Recovery phase. All the versions try to approach the following idle behavior when a packet loss is detected via Fast Retransmit:

- Divide the window by two.
- Retransmit as fast as possible all the packets that have been lost between the transmission of the lost packet and the detection of its loss.
- While recovering from losses, maintain a number of packets in the network equal to the new network capacity estimate.

- When all losses are recovered, get out of Fast Recovery into congestion avoidance. The network should contain at this point W_{th} packets. If it is not the case, the proposition [64] is to use slow start until reaching this number.

The different losses in the same window (i.e., between the first loss and the reaction of the source) should be considered at the source as the result of the same congestion event and the congestion window should be divided once by two. We will explain later how well the different versions of TCP approach this idle behavior.

In addition to the different algorithms for changing the window size, TCP congestion control contains other mechanisms for setting the retransmission timer [75, 108]. Finding the appropriate value for the retransmission timer is an important issue since it determines how correctly the source infers network congestion and how fast it reacts to this congestion. Whenever the source receives a new ACK that slides the left-hand side of its window (Figure 1.2), the retransmission timer is rescheduled with a value calculated from the average (SRTT) and the mean deviation (RTTVAR) of the round-trip time samples seen so far. The use of the mean deviation has been introduced in [75] and has been shown to be necessary to avoid wrong timeouts caused by round-trip time oscillations[10]. When a timeout occurs, the source retransmits the first unacknowledged packet and reschedules the retransmission timer with twice its current value. This is called the exponential back off of the timer [75]. An upper bound exists for the timeout duration (e.g., 64s) as well as on the number of successive timeouts for the same packet. The exponential back off of the retransmission timer in case of sever congestion is considered [75] as an important action for network stability.

2.2 TCP and large bandwidth-delay product networks

The bandwidth-delay product has considerably increased the last decade due to the expansion of the Internet and the increase in transmission rates. This has caused many problems to TCP. We will cite in this section the different mechanisms that have been proposed to help TCP to utilize efficiently the bandwidth available in today's networks.

The first problem that TCP encounters on large bandwidth-delay product paths is that the window must be able to reach large values in order to use efficiently the available bandwidth. But, the congestion window cannot exceed the receiver window which in turn is limited to 64 Kbytes (16-bit field, see Figure 2.1). This problem has been solved with the introduction of the window scale TCP option [77][11]. This new option carries a scale factor that the source multiplies by the old window field to get windows up to 2^{30} Bytes.

Another problem with operating at large windows is that a congestion event may lead to

[10] A wrong timeout is one that appears when the source under-estimates the round-trip time. The acknowledgement arrives after the expiration of the timer which could be avoided if the timer were correctly set.

[11] A TCP option is a set of fields attached to the header of a TCP packet. The source and the receiver agree on the addition and the structure of these fields upon connection set-up.

the loss of multiple packets from the same connection. An efficient Fast Recovery phase is then required in order to correct multiple losses in the same window and to avoid as much as possible the timeout and the slow start phase.

At large bandwidth-delay product, network buffers also have an important impact on the performance. These buffers must be well dimensioned and must be large as well. But, large buffers mean long queueing time which is undesirable for delay sensitive applications and short TCP transfers. We wish to keep the queueing time in network routers small especially on paths where the propagation delay is important. Indeed, for the same bandwidth-delay product, the queueing time increases with the propagation delay since the bandwidth is then smaller and more time is required to transmit packets. Some limitations may also exist on the buffering capacity of network routers (e.g., on board of satellites) which urges for mechanisms that reduce the buffering requirement. We will address all these issues in the following subsections.

2.2.1 Fast Recovery

Fast Recovery is called when a loss is detected with Fast Retransmit. Some ACKs (duplicate ACKs) still arrive at the source indicating that some packets are still reaching the destination. Fast Recovery tries to use the information carried by ACKs to estimate the number of packets in flight while recovering from losses. New packets are sent if this number falls below the network capacity estimate. This keeps the network well utilized and the ACK stream active so that timeout and slow start would be avoided. The network capacity is estimated after the loss detection to half the window size when Fast Recovery is called. The injection of new packets continues until all the losses in the same window of the first loss are recovered. Here, the source leaves the Fast Recovery phase and resumes its transmission in the congestion avoidance mode. Ideally, the source must have W_{th} packets in the network when Fast Recovery ends. It happens that due to the loss of ACKs on the return path, the source finds that it has less than W_{th} packets in the network when it exits Fast Recovery. The solution here [64] to avoid a burst of packets is to set the congestion window to the number of packets the source finds in the network and to use slow start until reaching W_{th}.

The difference between the different versions of TCP which implement Fast Recovery is in the way they estimate the number of packets in flight during the recovery phase. The Reno version of TCP [121] considers that every duplicate ACK is an information that a packet has left the network and it injects a new packet if the window allows. The problem with Reno, which is widely deployed in operating systems, is that it supposes that only one packet is lost upon congestion and leaves Fast Recovery when an ACK for the first loss is received. Leaving Fast Recovery early prohibits the source from detecting more than two losses in the same window with Fast Retransmit [56]. The number of new packets sent during recovery is too small so that three duplicate ACKs will not be received to trigger Fast Retransmit. The network drains, the ACK stream stops and a timeout is required to detect the other losses. Moreover, the slow start

phase after the timeout is called with a small threshold due to the division of W_{th} by two at every loss detection. In the case of multiple losses per window, it has been shown that Reno performs worse than the Tahoe version which always resorts to slow start [56]. The Reno version must be seen as a first and important step towards an efficient Fast Recovery phase.

New-Reno [64, 72] has been proposed to overcome the problems of Reno when multiple packets are lost in the same window. The idea is simple and consists in staying in Fast Recovery until all the losses in the same window are recovered. If all ACKs arrive at the source, the New-Reno version is able to keep W_{th} packets in the network while recovering from losses. This avoids the timeout and the slow start but cannot result in a recovery faster than one loss per round-trip time. This is simply because cumulative ACKs used by TCP cannot inform the source of more than one loss per round-trip time. The source needs to wait for the ACK of the retransmission of a loss to discover the next loss in the same window. Moreover, relying on ACKs to estimate the number of packets in flight leads to a problem when ACKs are lost on the return path. The loss of ACKs results in an underestimation of the number of packets that have left the network, thus in a small number of packets transmitted and a low utilization of the available bandwidth during the recovery. A smaller number of packets than what we intend (i.e., W_{th} packets) is kept in flight.

More information is needed at the source to recover faster than one loss per round-trip time and to estimate more precisely the number of packets in the pipe. This information is now provided by *SACK* (Selective ACK) [91]. SACK is a TCP option containing the three blocks of contiguous data most recently received at the destination. With this information, the source is able to detect more than one loss in a single round-trip time. Also, it can calculate more precisely the number of packets that have left the network without relying on the ACK stream. Many algorithms have been proposed to use this information during Fast Recovery. We find TCP-SACK [56] that uses ACKs for the estimation of the number of packets in the network as Reno, and SACKs to retransmit more than one loss per round-trip time. This leads to an important improvement in the performance when bursts of losses appear in the same window. But, the recovery is always sensitive to the loss of ACKs. As a solution to this latter problem we find FACK (Forward ACK) [90] that relies on the SACK information in the estimation of the number of packets in the pipe. Thus, FACK resolves the sensitivity of TCP-SACK to the loss of ACKs on the reverse path. The number and the identities of packets to transmit during the recovery phase is then decoupled from the ACK stream[12]. But, ACKs are still used to trigger the transmission of TCP packets. Later, we will see another proposition to decouple another mechanism of TCP from the ACK stream. The aim of all these works is to make TCP operation insensitive to any disturbance of ACKs on the reverse path. Such disturbance is harmful for TCP mechanisms that use the flow of ACKs to infer what is happening to data packets in the

[12]With Reno and New-Reno, both the number and the identities of packets to transmit during Fast Recovery are coupled with the ACK stream. With TCP-SACK, only the number of packets to transmit is coupled.

forward direction (e.g., Fast Recovery).

2.2.2 Buffering requirement in network routers

The size of buffers in network routers must scale with the bandwidth-delay product. These buffers serve to two main things:

- Absorb bursts of packets that arrive at a router at a rate higher than the rate of its output interface. This absorption should continue whenever the output link in not fully utilized.
- Provide a certain backlog of packets in routers that ensures the well utilization of the available bandwidth when the TCP sources back off their transmission rates.

But, the buffering must be limited to a small number of packets otherwise the end-to-end delay will reach high values and delay-sensitive applications (e.g., Telnet) will suffer. A certain tradeoff exists between increasing the buffer size to realize the above two objectives and reducing it to reduce the end-to-end delay. As we will see later, the buffer management technique plays an important role in this tradeoff. Such techniques indicate how to share the buffer size between the different flows, when to drop a packet, and from which flow to drop it. Given that TCP flows react to packet losses, changing the way with which packets are dropped changes the reactions of TCP flows which in turn changes the performance that we could obtain with a certain buffer size.

In the rest of this section, we explain how the buffer in a router must be dimensioned and the problems that result from an under-provisioning of buffers. We consider for this purpose the two modes of a TCP connection: the slow start mode and the congestion avoidance mode. First, the buffering requirement is studied under the classical drop-tail policy. For the first mode, we explain the problem of TCP burstiness and the required buffer size to absorb these bursts [84]. This analysis will be the basis of our work in Chapter 7 [29] on improving the performance of TCP in large bandwidth-delay product networks. For the second mode, we study the required buffer size to provide a sufficient backlog that ensures a full utilization of the available bandwidth. We explain the problem of synchronization of congestion control of TCP connections sharing a common bottleneck and its impact on the utilization [65, 128]. We then sketch about some buffer management techniques proposed to alleviate this problem of synchronization and to improve the utilization for a given buffer size.

Buffering requirement during slow start

The exponential growth of the congestion window during slow start results in bursts of packets sent at a rate exceeding the bottleneck bandwidth. Without loss of generality, suppose that the destination acknowledges all data packets. The case of other acknowledging strategies will be studied in Chapter 7 when we present our model for the variation of the queue in the bottleneck router during the slow start phase. ACKs then return to the source at the rate of the bottleneck.

Every ACK increases the congestion window by one segment and triggers the transmission of a burst of two packets. This will result in the source transmitting long bursts of packets with the rate of packets within each long burst equal to twice the bottleneck rate and with a frequency of one long burst per round-trip time [84]. A long burst corresponds in fact to a window size of packets. For explanation see Figure 7.2 for how packets circulate in the network in long bursts. These bursts are unavoidable since the current implementations of TCP don't provide any kind of spacing between packets. Sometimes the size of a burst is limited [56] but in general when the source decides to transmit multiple packets in response to an ACK, it sends them in a single burst at the maximum rate allowed by its output interface whatever is the bottleneck bandwidth. Arriving at the input of the bottleneck link, packets sent in bursts must wait till their predecessors get served. A queue builds up in network routers during slow start even if the congestion window is still smaller than the bandwidth-delay product. The length of this queue increases with the increase in the bandwidth-delay product of the connection's path. If the buffers are small, an overflow will occur during slow start before reaching the network capacity. We call this phenomenon an *early* buffer overflow [29]. It is early because an overflow should normally occur when the pipe between the source and the destination is filled not before as with this overflow. The TCP source considers here that the network is congested and reduces its slow start threshold which deteriorates the performance since the capacity of the network is underestimated. Slow start fails in this case in smoothing the transmission rate. A slower window increase is necessary to avoid the early overflow. But, a slower window increase means a longer slow start phase. This raises another tradeoff of TCP congestion control, that is the choice of the window increase rate. We will try to investigate this tradeoff in Chapter 7. We have to retain here two things. First, TCP is bursty for any window increase rate (even during congestion avoidance) and the burstiness (i.e., the ratio of the transmission rate of TCP and the rate of the bottleneck) increases with the acceleration of the window growth. Second, for any window growth rate, there exists a router buffer that is not able to absorb the resulting burstiness. The buffering requirement in the router increases with the window growth rate. It is approximately negligible during the congestion avoidance mode as we have shown in [28] (a buffer equal to one packet). It could be important during slow start especially if the bandwidth-delay product and the slow start threshold are large.

To illustrate this burstiness of TCP, we plot in Figure 2.3 the occupancy of the buffer in the bottleneck router while a Reno TCP connection progresses in the slow start mode (between 0 and 1.5s). We also plot in the same figure the variation of the bandwidth utilization. Bandwidth utilization is calculated by averaging the rate of packets over one round-trip time intervals. The utilization is multiplied by the buffer size to fit with the buffer occupancy on the same plot. The simulation scenario in Figure 7.3 is considered for this purpose (B=20 packets, ACKs delayed). We see clearly how a queue builds up due to slow start long bursts and how the buffer overflows before the full utilization of the bottleneck bandwidth.

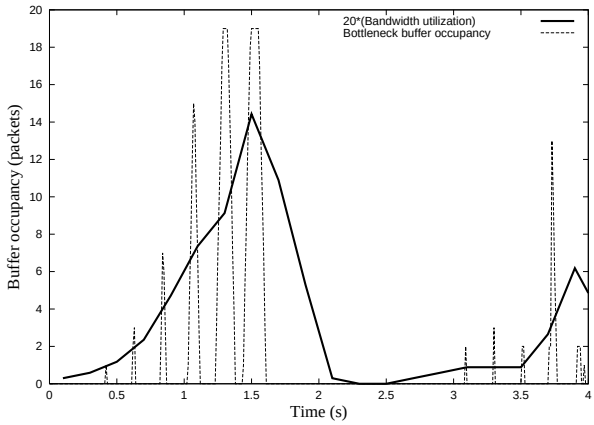


Figure 2.3: Queue length and utilization during slow start

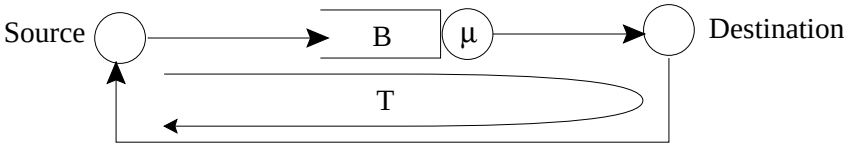


Figure 2.4: Single-node network model

The problem of early losses during slow start has been first studied in [18, 84]. The authors consider a long-life TCP Tahoe connection where slow start is called after every loss detection. They found a condition on the required buffer size to avoid an early buffer overflow in the stationary regime of the connection. They modeled the network with a single bottleneck node of bandwidth μ packets/s and of buffer size B packets (Figure 2.4). T represents the two-way propagation delay. They calculated the maximum window size of a Tahoe connection (i.e., the pipe size) as being $W_{max} = B + \mu T$. Slow start should then end at $W_{th} = W_{max}/2$. Given that packets are transmitted during slow start at twice the bottleneck rate μ, half the window size is supposed to be queued in buffer B. Thus, B must be taken larger than $W_{th}/2$ in order to avoid an early buffer overflow. Using that $W_{th} = (B + \mu T)/2$, this gives a buffer size larger than $\mu T/3$ (one third the bandwidth-delay product).

The analysis in [18, 84] is not applicable to the other versions of TCP (e.g., Reno [56]) where slow start is only called at the beginning of the connection and after a timeout. This is simply because W_{th} is different. For example, in case of timeout, the slow start threshold might be divided multiple times by two resulting in a less important buffering requirement. For the slow start at the beginning of the connection, the required buffer size depends on the value we give to W_{th}. For a slow start threshold equal to the bandwidth-delay product as proposed in [72], the router must be able to queue more than half the bandwidth-delay product to avoid early losses. For a slow start threshold set to larger than the network capacity ($W_{th} > W_{max}$), a buffer size

larger than the bandwidth-delay product is required so that TCP can fill the network capacity during slow start. These numbers are given for a receiver that acknowledges all data packets. A thorough analysis of the interaction between slow start (or window increase rate) and buffer size is presented in Chapter 7.

We still have to say some words about the single-node network model used to derive conditions on B. This model makes the assumption that packets are queued in only one router which is the slowest one on the path of the connection. This is true if all upstream routers are too fast compared to the bottleneck. However, we have shown in [28] that if one of the upstream routers has a rate less than the transmission rate of TCP during slow start (i.e., twice the bottleneck rate if all packets are acknowledged), packets will be also queued in this router. In this case, the burstiness of TCP will be absorbed by both routers not only by the bottleneck one. Thus, the upstream routers must be also well dimensioned. In [28], we derived conditions on how to dimension the buffer in the bottleneck as well as in upstream nodes. We proposed an equivalent model of two-nodes that gives the same performance for TCP when used instead of a network of multiple routers in tandem. The second node is the same as the one in the single-node model. The first node represents all upstream routers that may participate in the absorption of TCP burstiness. Later, we assume that upstream routers are quite fast so that a single-node model for the network works correctly. This can be seen as the worst case when the bursts of TCP are only absorbed by the router we focus on.

Buffering requirement during congestion avoidance

In this mode, packets on a TCP connection are transmitted at approximately the bottleneck rate. A simple calculation [28, 29] (one can use our result in Chapter 7) shows that the rate of TCP packets within long bursts is equal to $(bW+1)/(bW)$ the bottleneck rate, where W is the current window size and b is the number of data packets covered by an ACK. Thus, the burstiness of TCP is easily absorbed and a queue starts to build up in routers only when the window exceeds the bandwidth-delay product (μT in Figure 2.4). In contrast to slow start where congestion may occur early, congestion occurs during congestion avoidance when the pipe between the source and the destination is filled. The source then divides its window by two and starts a new congestion avoidance cycle, of course if the Fast Recovery algorithm works as desired. To get a throughput approximately equal to the bottleneck rate (i.e., a full bottleneck utilization), the new window after the reduction must be larger than the bandwidth-delay product. This requires a network pipe ($B + \mu T$) in Figure 2.4 larger than twice the bandwidth-delay product (μT), hence a large buffer size at least equal to the bandwidth-delay product. This important buffering requirement is the normal consequence of the conservative reaction of TCP (division of the window by two). In reality, the conservative reaction of TCP was the result of the large buffers that were deployed in routers when congestion control has been added to TCP [75]. It was observed that a conservative reaction is required to reduce the number of queued packets.

But, with the increase in link rates and with the expansion of the Internet, the queueing of a bandwidth-delay product of packets has become a problem. First, it means a storage of a large number of packets which can be difficult in some environments due to memory constraints (e.g., on board of a satellite). Second, the queueing of a bandwidth-delay product of packets means a queueing time equal to the propagation delay which is unacceptable for some delay sensitive applications (e.g., Telnet). Clearly, the second problem is more pronounced on long delay paths. Hence, a tradeoff exists between increasing buffers to improve the utilization and decreasing them to deliver quickly TCP packets.

The previous analysis holds in the case of a single connection. However, when multiple connections share the same bottleneck, the decrease in the load at the moments of congestion should be less than that in the case of a single connection; ideally, only a small number of connections should reduce their windows and those are the connections that consume the maximum bandwidth. Given that the window size of a connection is much smaller than if a single connection shares the bottleneck bandwidth, the decrease in the total load is less important. Hence, the presence of multiple connections should smooth the load on the network and reduce the buffering requirement. Unfortunately, this is not the case with classical drop-tail routers. A congestion event causes losses from most of the connections (if not of all of them) and the load is again divided by approximately two as in the case of a single congestion. This is the famous problem of *synchronization* of TCP congestion control [46, 65, 84, 128]. It can be explained as follows. A drop-tail router discards packets only when the buffer is full. A complete round-trip time is required between the drop of a packet and the reaction of the source to which the packet belongs. During this round-trip time, the bottleneck router remains congested and drops all packets exceeding the network capacity. Those packets belong to the different connections that increase their windows during this round-trip time. Hence, multiple connections divide their windows at the same time which results in an important decrease in load until the next congestion event. Due to this synchronization, the presence of multiple connections does not reduce the buffering requirement and a buffer size equal to the bandwidth-delay product is still required to achieve full bottleneck utilization.

The solution to this buffering requirement problem has been brought by active queue management buffers [42] and namely by Random Early Detection (RED) ones [65]. The main idea behind these queues is to solve the problem of synchronization by distributing packet drops over time, not to wait until the last minute and drop a large number of packets as with drop-tail buffers. Distributing drops over time will decrease the amount by which the load is reduced upon congestion since the connections will no longer reduce their windows simultaneously. This makes the load in the network more uniform (i.e., less oscillatory) and the number of packets to queue in routers smaller. The queues in network routers will oscillate more frequently (due to the distribution of drops) but with smaller variations and a smaller average value. Ideally, the queue oscillations should disappear when the number of connections becomes very large.

Active queues consist of three main algorithms which, combined together, lead to the desired behavior. The first algorithm consists in anticipating the congestion by setting some thresholds lower than the buffer size. The congestion is considered as not severe when (some averaged version of) the queue exceeds a minimum threshold and it is considered as severe when the queue exceeds a maximum threshold. An active queue is supposed to operate between the two thresholds. The second algorithm consists in dropping packets randomly when the congestion in the network is not severe and dropping all packets as a drop-tail queue when the congestion is severe. The third algorithm consists in absorbing the oscillations of real queue by some averaging algorithm. An Exponentially Weighted Moving Average algorithm is used for this objective. The average queue is used in the decision on the congestion level and in the calculation of the probability with which incoming packets have to be dropped. This probability is an increasing function of the average queue. If well designed, the oscillations in the real queue should be completely absorbed which permits a constant average queue and hence a dropping of all packets with approximately the same probability. This yields the best distribution of congestion events over time and thus the best uniformization of load and queues.

Although they bring a solution in the case of multiple connections, active queues worsen the situation in the case of a small number of connections. The use of a low threshold causes an important decrease in load as if a small buffer is used. In fact, active queues has been designed for backbone routers where a large number of connections exist at the same time. With the current conservative reaction of TCP sources, it seems impossible to find a solution to the problem of utilization in the case of a small number of connections without queueing a large number of packets in routers. The number of packets to queue depends on what we intend to optimize: the end-to-end delay or the utilization. A possible solution could be to scale the thresholds of the active buffer queue as a function of the number of connections. Another possible solution could be to change the reduction factor of TCP congestion control so that the load does not drop drastically when a congestion occurs. This latter solution will reduce considerably the buffering requirement. The Vegas version of TCP [45] works in this direction. It solves this dependency of the utilization and the delay on the buffer size by not basing the congestion control only on losses. The increase in the round-trip time is used to maintain the window at a size equal to the bandwidth-delay product plus some packets (between 1 and 3) in the bottleneck router. Except when the congestion is inferred from the loss of packets, the window is reduced by one packet rather than by half. Thus, the utilization is most of the time close to the bottleneck bandwidth and the end-to-end delay is kept close to the propagation delay. This is in theory how TCP Vegas should operate. The reality has shown [38, 88, 116] that the variation in round-trip time is not a reliable information for inferring network congestion. An explicit feedback from the network is required if we want the TCP source to react to different levels of congestion with different reduction factors.

Before ending this section, we will refer to other simple buffer management techniques that

can alleviate the problem of synchronization and reduce the buffering requirement. Recall that the synchronization is the result of the drop-tail policy. Thus, dropping a packet from the queue when a new packet arrives and finds the queue full will alleviate the problem. One could drop the packet at the head of the queue [86] or that at a random position [69]. The number of packets dropped at the moment of congestion remains approximately the same as in the case of a drop-tail buffer. The difference is that these drops will be distributed on a small number of connections (those having a large number of packets in the buffer at the moment of congestion) and not on all the connections. The decrease in the load is then smaller which should improve the utilization for a given buffer size. This will reduce the required buffer size to achieve a certain utilization.

2.3 TCP and long round-trip times

The geographical expansion of the Internet and the introduction of satellite networks have led to paths of long round-trip times. For example, the one-way propagation delay across a GEO satellite is in the order of 250 ms. The first problem with these long delay paths is not particular to TCP but it is common to any other closed-loop congestion control mechanism. The reaction of TCP to network congestion takes effect one round-trip time later. At long round-trip time, this reaction may become unnecessary or insufficient which results in an inefficient utilization of network resources. The other problems however are proper to TCP and are due to the fact that the window increase is only a function of the number of ACKs received and not the round-trip time. This results in a window increase rate inversely proportional to the round-trip time. The longer the round-trip of the connection, the longer the time the source requires to pass from a window size to another. In principle, this has no great impact on the congestion avoidance phase since the source is supposed to operate in this phase at a large window. However, this has a large impact on the slow start phase which is a transitory phase designed to fill quickly the network capacity. The slow start phase will last longer. Given that the network is underutilized during slow start due to the small size of the window, the performance of TCP degrades [3, 7]. This degradation is more important in case of short transfers such as Web transactions. These transfers complete in general in slow start and suffer from a long delay and a low throughput. Accelerating the window increase during slow start without overwhelming network buffers has been considered in the literature as the most important issue for improving the performance of short TCP transfers in a satellite environment [3].

Another problem caused by the dependency of the window increase on the round-trip time is the *unfairness* in the allocation of the bottleneck bandwidth between connections of different round-trip times. As we explained before, this is due to the fact that the different connections increase their windows at different rates [50]. Many works have shown the bias of TCP against connections with long delay paths [60, 71, 84]. Short delay connections increase their rates more

quickly and grab most of the available resources. In case of drop-tail buffers and in presence of the synchronization phenomenon, the throughput of a connection has been shown [84] to be inversely proportional to the square of its average round-trip time. We can say that it is inversely proportional to T^{α}, where T is the two-way propagation delay and α a factor between 0 and 2. The factor α is the result of all the connections having approximately the same queueing time in the bottleneck router [84]. The larger the buffer size at the bottleneck, the smaller the factor α. Ideally, when the buffer size in infinite, all the connections should get the same throughput whatever are their propagation delays. We will only focus in this section on the unfairness of TCP caused by round-trip times. Another unfairness problem has been addressed in the literature, that caused by the difference in the number of routers crossed by the connections [60, 74]. A TCP connection crossing a large number of routers has been shown to get less throughput than a connection crossing a small number of routers. This is simply because the probability that a packet is lost is larger in the first case. In our study, we assume that connections cross the same number of congested routers and hence they have to realize the same throughput whatever are the resources they consume in other parts of the network. We present the different schemes proposed in the literature to achieve this fairness objective. We don't address the case of different congested routers where other fairness objectives could be envisaged [89]. The fairness objective we consider is often called in the literature the *max-min* fairness and it briefly consists in always helping the TCP connection with the lowest throughput.

2.3.1 Proposed solutions to accelerate the window increase

Many propositions have been made to accelerate the window increase at the beginning of a TCP connection. These propositions try to reduce the number of round-trip times required to reach the network capacity estimate. In the absence of any kind of packet pacing, this problem seems to be unsolvable on an end-to-end basis. We must reach as fast as possible a certain window (W_{th}) but we are only allowed to transmit packets in bursts. The acceleration of window growth increases the burstiness of the protocol which overloads the network routers and impairs the performance of the connection and that of other connections. Note here that the problem of slow window increase on long delay paths is exacerbated when ACK are lost or when they are delayed at the receiver. The proposed solutions to this problem can be divided into three categories: some of them change the window increase algorithm of TCP. This is the kind of solutions that we will study in Chapter 7. Other solutions solve the problem at the application level and the others solve it inside the network.

TCP-level solutions

The first proposition was to use a window larger than one segment at the beginning of slow start [3, 6]. This large window has been proposed for the first slow start phase not those following a timeout. After a timeout, the network is supposed to be severely congested so that a

transmission at a window of one packet is required. An initial window of maximum 4 segments has been proposed [6]. Another proposition, called Byte Counting, consists in accounting for the number of bytes covered by an ACK while increasing the window [2, 3]. The congestion window is increased as a function of the number of data packets acknowledged rather than the number of ACKs received at the source. The main motivation behind this solution was to recover from the negative impact of the delay ACK mechanism [41] on TCP performance in a satellite environment. It decouples the window increase algorithm from the ACK clock. We already saw how the SACK option [91] decouples the Fast Recovery algorithm from the ACK clock. But, Byte Counting may result in bursts when an ACK covers many packets as when doing a slow start after a timeout period. For this reason, a limit has been proposed on the amount by which an ACK can increase the window (Limited Byte Counting) [2]. In addition to decoupling the window increase from the ACK clock, the Byte Counting solution can be seen as an attempt to accelerate the window increase. It improves the performance in some situations. However, it increases the burstiness of slow start which may deteriorate the performance if network routers are not well dimensioned. In chapter 7 we study the problem of slow start burstiness for any window increase policy including those proposed by standard TCP and Byte Counting. We define the optimal policy for window increase and based on it, we present an improvement of Byte Counting we call the *Decreasing Byte Counting* solution.

The natural solution to the problem of TCP on long delay paths is to eliminate the burstiness of the protocol by pacing packets at the rate of the bottleneck. This will also form a solution to the problem of large buffering requirement during slow start that we already studied. It is proposed [9, 19, 72] to use the flow of ACKs at the beginning of the connection to get an estimate of the bottleneck rate. Once this estimate is obtained, the source can set its slow start threshold to an appropriate value (to the bandwidth-delay product for example) and can transmit directly W_{th} packets at the bottleneck rate. Once the slow start threshold is reached, the source switches to a normal congestion avoidance phase. This should shorten the slow start phase while not overloading the network. An important gain in performance has been noticed with this solution [19]. The question here is how reliably we can estimate the bottleneck rate from end-to-end measurements. Another question is how much a jump of the window impacts the performance of other connections using classical slow start. There is also the question of the load caused by fine granularity timers required for packet pacing at the TCP source. Pacing packets seems to be the best direction for solving all the problems related to the slow start phase of TCP, but more works are still required for such direction to be adopted.

Application-level solutions

These solutions try to solve the problem of long slow start duration at the application level without changing TCP. The application manages the data to transfer and the establishment of TCP connections in a way to improve the overall performance. The first solution, called

XFTP [8], consists in establishing multiple simultaneous TCP connections for the transfer of a single file. This should improve the performance since the increase rate of the overall window is larger. Also, the distribution of losses on multiple connections instead of one makes the error recovery algorithm of TCP more efficient and the decrease in load upon congestion less important. However, this solution increases the aggressiveness of the transfer and hence the losses in the network. An adaptive mechanism has been proposed [8] to change the number of connections as a function of network congestion. Such solution can be seen as an unsocial behavior where a user cheats the network to get the rate of multiple users.

Another solution has been proposed to accelerate the transfer of WEB pages. It can be also used for any other application desiring the download of a set of small files from the same server. The objective of this solution is not to accelerate the slow start phase of a transfer as with the previous solution, but rather to optimize the transfer of the set of objects in a WEB page (text zones, images) on a single TCP connection. Instead of using an independent TCP connection to fetch every object (which is so costly since the objects are in general of small size and since a connection establishment procedure and a slow start phase are required for every object), the client establishes a *persistent* connection and asks the server to send all the objects on it (HTTP 1.1 [57]). With this persistent connection, only the first objects of the page suffer from the long slow start phase, but once slow start is terminated and the connection passes to congestion avoidance, the remaining objects are transfered at a high rate. Hence, the low rate during slow start is compensated by the long time the connection stays in the congestion avoidance mode. Recently, a proposition has been made [54] to accomplish this aggregation of transfers at the TCP level rather than at the application level. The advantage of such aggregation is that it can be used by different applications of different versions, not only HTTP. Applications continue to establish a TCP connection per object. At the TCP level, connections to the same destination are grouped together and their congestion windows are changed in a way that their sum behaves as that of a single TCP connection. When a new object is to be transfered, the corresponding connection starts with a window that depends on the priority of the object, and this is without changing the total window of the group. One possible scenario is to divide the total window of the group equally between the different connections. The first packets of the object are paced at a rate calculated from the associated window and from the average round-trip time of the group.

As a further application-level solution, we find the caching (i.e., the use of a proxy server) [129]. The TCP connection initiated by the client is established to a nearby cache which maintains a copy of recently downloaded files. If the requested file exists in the cache, it is directly delivered to the client which results in a short transfer time. If not, the file is downloaded from the server (or from a higher-level cache), stored in the cache for future requests, and then delivered to the client. If the caching policy is appropriately tuned, this should accelerate slow start and improve the performance of most of the transfers. Note here that caching is better supported by satellites due to their broadcast nature. Once a cache searches a file, all the caches in the satellite scope

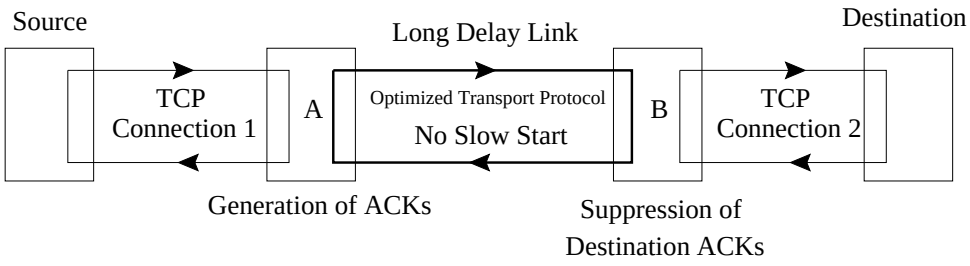


Figure 2.5: TCP-Spoofing: elimination of the long delay link from the feedback

receive it.

Network-level solutions

This approach consists in solving the problem of slow start inside the network rather than in hosts. It is worthy when a long delay link (e.g., a satellite link) is located on the connection's path. In order to decrease the round-trip time of the connection, the long delay link is eliminated from the feedback loop by acknowledging packets by the router at the input of this link (A in Figure 2.5). The packets are then transmitted on the long delay link (A-B) using an optimized transport protocol (e.g., the STP protocol proposed in [70]). A modified version of TCP can be also used [94]. Given that the bandwidth between the two ends of the long delay link (between A and B) and the buffering capacity at its input (at A) are known, the transport protocol to be used on the long delay link can be tuned in order to increase quickly the transmission rate without the need for a long slow start phase. Once arriving at the output of the link (at B), another TCP connection is used to transmit the packets to the destination. The long delay link may lead directly to the destination as when a satellite network is used to access the Internet. The second TCP connection is not required in this latter case. Now, given that packets are acknowledged by the router at the input of the long delay link (A), any loss between the input of the link (A) and the destination must be retransmitted on behalf of the source. Also, ACKs from the receiver must be silently discarded (on B) so as not to confuse the source. Such solution is called *TCP-Spoofing* since the source is spoofed by the router at the input of the long delay link.

The main gain in performance with Spoofing comes from not using slow start (or from using a faster version) on the long delay link. This, together with the fast increase in the congestion window on both sides of the long delay link, improves the performance of the transfer. Another advantage of Spoofing is that it shortens the feedback loop of TCP and improves its reaction to network conditions. A similar solution has been proposed in ATM networks for the improvement of ABR rate control over long delay paths [21]. A long end-to-end ABR connection is divided into small ones. The switch at the input of a small connection behaves as a Virtual Source (for example, node B in Figure 2.5), and the switch at its output behaves as a Virtual Destination. A

virtual destination returns a feedback (Resource Management cells) to the virtual source located before it which in turn controls the rate of ATM cells as a function of the available bandwidth between them. This is claimed [21] to improve the overall rate control. As we will see later, splitting a TCP connection into small ones also results in a better resilience against losses not caused by congestion.

Despite these advantages, Spoofing still has a lot of drawbacks. First, it violates the end-to-end semantics of TCP. A packet is acknowledged (by A) before reaching its destination. Also, it does not work when encryption is accomplished at the IP layer and it introduces a heavy load on intermediate routers (A and B). Furthermore, the transfer is vulnerable to path changes and ACKs have to follow the same path as data packets (of course in an opposite direction) in order to be discarded by intermediate routers before reaching the source.

2.3.2 Proposed solutions to improve the fairness

TCP is known to be unfair against connections with long round-trip time [60, 84]. Two trends exist to improve the fairness of the protocol. The first trend tries to solve the problem at the TCP level by accelerating the window growth for long delay connections. The second trend keeps TCP unchanged and tries to solve the problem inside the network. Some mechanisms are proposed for network routers so that to allocate the available bandwidth fairly between different connections.

As TCP-level solutions, we find first the proposition made in [60] and called *Constant Rate* algorithm. The author proposes to increase the window W in the congestion avoidance mode by a factor proportional to the inverse of the square of the average round-trip time (RTT):

$$\text{When a non-duplicate ACK is received: } W = W + c/[W(RTT^2)].$$

Note that current versions of TCP increase the window by $1/W$ when a non-duplicate ACK is received [75]. The result of this new algorithm is a constant increase rate (equal to c) of the connection transmission rate regardless of the round-trip time. According to [50], this should lead to fairness. Recall that current versions of TCP are not fair since they increase their transmission rates with a slope inversely proportional to RTT^2. The first problem with this new algorithm is the choice of the factor c. Second, accelerating the window increase while preserving the ACK clock results in large bursts for long delay connections. In [71], the authors show that with drop-tail buffers, this increase in burstiness may impair the fairness instead of improving it. To solve the problem, they propose instead an *increase-by-K* algorithm: the window is increased by K segments every round-trip time and this is only for long delay connections. The performance of long delay connections improves without affecting that of the others when K is given small values.

Now, in order to improve the fairness inside the network without changing TCP, flows of different connections must be isolated from each other in network routers. Given that congestion

control in TCP is based on losses, isolation means that a congested node must distribute losses between the different TCP connections in such a way that they get the same throughput. The best isolation is achieved by accomplishing a per-connection queuing with a Round Robin service discipline [120]. Such solution ensures fair share of bandwidth at short time scales but it is very hard to implement given the large number of TCP connections that cross an Internet router. An easier solution for implementation is to serve all packets in a FIFO (First In First Out) manner and try to ensure fairness by managing intelligently the buffer space. With the FIFO service discipline and on long time scales, the bandwidth of a link and the buffer at the input of the link are shared in the same way between the different connections. Many buffer management policies have been proposed. Some of these policies, as ERD (Early Random Drop) [69] and RED [65], choose to drop incoming packets with a certain probability when the queue length or its average exceed a certain threshold. The aim is to distribute losses on the different connections proportionally to their rates. These policies improve the fairness while not requiring any per-connection state. As we will see in the next chapters when we address the end-to-end modeling of TCP, dropping packets of different connections with the same probability makes the throughput inversely proportional to RTT instead of being inversely proportional to RTT^2 in case of classical drop-tail buffers. Other works [87, 120] suggest that dropping packets proportionally to the rate of flows is not sufficient for isolating TCP flows from each other or from other non-TCP flows. Given that packets are dropped randomly, it is possible that a connection with a small window loses its packets which pushes her into a sequence of timeouts. Also, it is possible that a non-TCP flow not sensitive to packet losses grabs most of the bandwidth and the buffer space, and shuts down all the TCP connections. These latter works conclude that a better isolation between flows requires a fair sharing of the buffer space between different connections [87, 120]. Aggressive connections must be prevented from monopolizing the buffer space and a connection with a small window must be guaranteed to find some places in the buffer. We find here the other set of buffer management policies [68, 87, 120] (Flow RED, Fair Buffer Allocation, Virtual Queueing, Longest Queue Drop, etc.) that try to achieve fairness by sharing the buffer space fairly between connections. A long delay connection is now sure to find some places in the buffer which permits it to proceed without being shut down by short delay connections. Note here that the overhead from fair buffer sharing is less than the overhead from per-connection scheduling since we only need to account for the connections that have packets in the buffer not for all the active connections. Clearly, such overhead is more than the overhead of policies that don't consider buffer sharing (e.g., drop-tail, RED, etc.).

Solving the fairness problem at the TCP level has the advantage of keeping routers simple. But, it is not enough in a network where non-TCP flows exist [62]. Some mechanisms inside the network are required to protect conservative TCP flows from aggressive ones. These mechanisms are also required to ensure fairness at a level above TCP, say at the user or at the application level. A user (e.g., someone running XFTP [8]) may cheat the network and establish multiple

TCP connections in order to increase its share of the bandwidth. In such situation, the total flow of packets generated by these connections should be considered by the network as a single flow, and the network should manage to divide fairly its resources between flows not connections. This requires an aggregation in flows of TCP packets crossing a router. The level of aggregation determines the level of fairness we want. The source and destination IP addresses, the transport protocol used, the source and destination port numbers, etc., could help the network in the identification of flows.

2.4 TCP and non-congestion losses

TCP congestion control is based on the loss of packets. This results in a severe throughput deterioration if packets are lost on the path of the connection for other reasons than congestion. Non-congestion losses involve all kinds of losses that are not caused by a sustained congestion of network routers and thus that don't require a reduction of the rates of TCP flows. Their negative impact on TCP performance increases with the bandwidth-delay product of the connection's path [84]. A non-congestion loss that occurs when TCP is transmitting at a large window reduces the load more than when TCP is transmitting at a small window. In the former case, the source requires more time to return to its rate before the reduction of the window.

Non-congestion losses are mostly caused by transmission errors. A packet may be corrupted while crossing a bad quality link (i.e., a link with a low signal to noise ratio). The corrupted packet is discarded at the link level, IP level, or TCP level without being handed to the application. Most of the corrupted packets are discarded at the link level, but it happens that some of them pass the link-level test and reach the higher layers where they are discarded [122]. The detection of corruption in a layer is done by recalculating a checksum (on all the packet or only on the header as in the IP layer) and comparing it to that carried in the packet header. The packet is discarded if checksums differ. Bad quality links involve mainly wireless links where many phenomena such as interference, absorption, obstacles and others [55], may reduce the power of the signal at the receiver and cause a misinterpretation of the arriving information. Wireless links are now widely seen in the Internet. We find them in satellite and terrestrial mobile networks. In mobile networks, the mobility of users induces other phenomena that increases the loss probability of a packet. Fading, shadowing, handover, etc. [106], may put the wireless link in a bad state for a non negligible time which results in bursts of losses.

In [84] losses due to a transient congestion of network routers are also considered as non-congestion losses. A transient congestion is defined as being a congestion event that lasts for a small time compared to the round-trip time of the connection and that disappears before a TCP flow reduces its rate. A TCP reaction to this short congestion is not necessary. Such losses can be caused by a low rate bursty source sharing the bottleneck with a TCP connection. Losses inside an ATM backbone providing an ABR service to carry TCP traffic [68] can be also considered as

transient congestion events. In this latter case, the rate of TCP should only be reduced when the buffer at the input of the ATM cloud overflows.

The solutions proposed to the problem of non-congestion losses can be divided into two main categories. The first category consists in hiding the lossy parts of the Internet so that only congestion losses are detected by TCP sources. These solutions require some work in the network but have the advantage of not requiring any change to TCP. They are compliant with current TCP congestion control which requires that the network only contains good quality links below the IP layer. The second type of solutions consists in enhancing TCP with some mechanisms to help it to distinguish between different the two types of losses. The latter solutions can be considered as an attempt to decouple congestion control of TCP from its error control.

2.4.1 Hiding lossy parts of the Internet

This set of solutions consists in recovering non-congestion losses locally without the intervention of TCP sources. Hence, the source sees a clean path and continues to increase its rate until a real congestion occurs. A local recovery can be accomplished at the link level or at TCP level. From IP point of view, a link is any transmission medium located between two adjacent routers. If a link-level solution is not possible (i.e., difficulty of the modification of the link layer protocol), a recovery at the TCP level can be used instead. A TCP agent in an intermediate router monitors TCP packets, detects losses and retransmits them on behalf of the source. This agent must deal with congestion control and resource sharing in the lossy part it is hiding from the TCP connection, otherwise it must be intelligent so as to only hide non-congestion losses and let the TCP source react to congestion losses. In general, the lossy part is the last hop to the destination and a local congestion control is not required.

Link-level solutions

Two well known mechanisms exist for the improvement of a link quality: Retransmissions (ARQ) and Forward Error Correction (FEC). ARQ is efficient when losses are not frequent and when the propagation delay is not important. An extra bandwidth is only consumed when a packet is retransmitted. However, link-level retransmissions may interfere with TCP mechanisms [26]. If the link layer does not provide an in-order delivery of packets to the IP layer, TCP packets following the loss reach the destination before the retransmission and trigger the transmission of duplicate ACKs. Duplicate ACKs may reach the source while the link layer is trying to retransmit the packet. This causes an unnecessary window reduction and an unnecessary retransmission of the lost packet at the TCP source. The proposed solution to this problem is to use a link layer protocol aware of TCP mechanisms (Figure 2.6). The link layer suppresses the duplicate ACKs (at router R) so that they don't reach the source. If the link layer fails to retransmit the packet, the source will timeout and retransmit the packet itself. Note here that this solution is only applicable when the lossy link is the last hop to the destination. If the lossy link is followed by

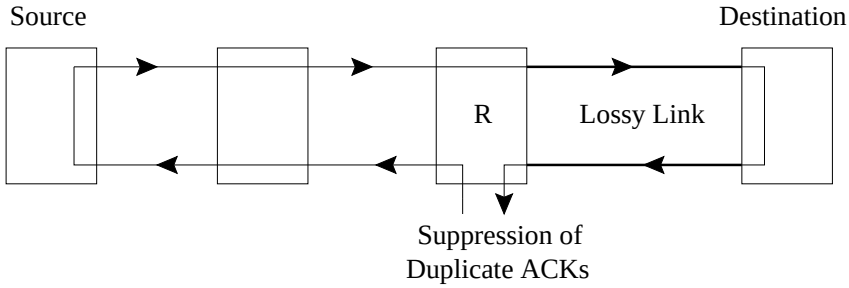


Figure 2.6: A TCP-aware link layer

other routers, congestion losses will be hided which we must not allow.

Now, FEC consists in sending in addition to the useful data, some redundant information in order to rebuild the corrupted part of the packet at the output of the lossy link. For example, one could send in addition to two blocks of data A and B, the result C of an XOR operation on these two blocks [119]. If either A or B is lost, the lost packet can be rebuild from an XOR operation on C and the other received packet. The drawback of this technique is that the redundant information is not used when the link is in a good state (e.g., when A, B and C are correctly received). Thus, it represents a certain waste of bandwidth. Also, the computation of the redundant information requires extra CPU processing time, memory and blocking delay. However, the advantages of FEC are in general worth the cost. First, corrupted packets are corrected runtime without any retransmission which is important for long delay links. Also, FEC does not interfere with TCP mechanisms. Packets are corrected on runtime without any increase in end-to-end delay and without any reordering of packets. For these reasons, this technique has been recommended in a satellite environment [7]. Convolutional coding, Viterbi decoding together with interleaving techniques and Reed-Solomon encoding, are widely used to render satellite links as clean as terrestrial ones. Given this importance of FEC for TCP congestion control, we will dedicate Chapter 9 to the evaluation of the gain in TCP performance for given FEC parameters. We will also study the variation in TCP/FEC performance when the state of the wireless link changes. We conclude our analysis in Chapter 9 by some guidelines on how to dimension FEC for a better TCP congestion control.

TCP-level solutions

These solutions propose to retransmit packets inside the network at the TCP level instead of the link level. The benefit is that we don't violate the principle of layering as with TCP-aware link layers. According to these solutions, a TCP agent in the router at the input of the lossy link keeps a copy of every data packet. It discards this copy when it sees the ACK of the packet and it retransmits the packet on behalf of the source when it detects a loss via duplicate ACKs. The retransmission is accompanied by a discarding of duplicate ACKs. This technique has been

proposed [24, 26] for terrestrial wireless networks where the delay is not very important to require the use of FEC. The TCP agent is placed in the base station at the entry of the wireless network. Two possible implementations of this agent have been proposed.

The first implementation (Indirect TCP [24]) consists in terminating (or splitting) the originating TCP connection at the entry of the lossy link. The intermediate agent then acknowledges the packets and takes care of handing them to the destination. Another connection well tuned to a lossy environment (e.g., TCP-SACK [56]) is established across the lossy part of the network. A different transport protocol can be also used across this lossy part. Such solution is similar to the Spoofing solution described in Section 2.3.1. It aims to eliminate the lossy link from the feedback loop, whereas with Spoofing the objective is to eliminate the long delay link. As with Spoofing, this solution violates the end-to-end semantics of the Internet. Also, it causes difficulties during handover since a large state must be transfered between base stations.

The second implementation, called *Snoop* protocol [26], respects the end-to-end semantics. The intermediate agent does not terminate the TCP connection. It just keeps copies of data packets without generating any artificial ACK. New ACKs sent by the destination are forwarded to the source. However, duplicate ACKs are stopped. A packet is locally retransmitted when three duplicate ACKs are received or when a local timeout expires. This local timeout is set of course to a value less than that of the source. The drawback of Snoop is that an interference may occur between the source and the agent mechanisms as in the link-level case. In fact, the Snoop protocol is no other than a link-level recovery implemented at the TCP level.

2.4.2 End-to-end solutions

Some authors have investigated the possibility of enhancing TCP congestion control to cope with non-congestion losses on end-to-end basis. They try to find a solution to the well known problem: how to decouple TCP congestion control from error control so that TCP will no longer use losses to infer network congestion. Two directions exist in the literature. The first one keeps TCP congestion control unchanged and tries to infer non-congestion losses at the source either via an explicit signal sent from the network or via the observation of round-trip time variations. The second direction consists in changing TCP congestion control so as to use other means for congestion detection. Even though the works in the second direction are not motivated by the problem of non-congestion losses, they will form a solution to this problem if they work correctly.

For the first direction, an *Explicit Loss Notification* (ELN) signal has been proposed [53, 116] to infer the source explicitly of the occurrence of a non-congestion loss. The source then reacts by retransmitting the lost packet without reducing its window. An identical signal has been proposed to halt the congestion control at the source when a disconnection appears due to handover in cellular networks. In [47], the problem of disconnection has been solved intelligently without the need for an extra signal. The sources are halted by setting to zero the window field in ACKs. The difficulty with ELN is that a packet corrupted at the link level is discarded before reaching

TCP and hence it is difficult to know about this corruption. Another problem with ELN is that the information in the header of a corrupted packet may not be valid, so the ELN signal may be sent to another source or may request the retransmission of another packet. It seems difficult to solve the problem of non-congestion losses with explicit signals. For this reason, it has been proposed in [38] to observe the variation of the round-trip time with the congestion window and to try to infer from this variation if a loss is caused by a network congestion or no. No explicit signals and no mechanisms in the network are required. Unfortunately, the results were negative and the distinction between the two types of losses was not possible. This simply because the variation of the round-trip time with the window has been shown not to be an efficient method for the anticipation of network congestion [88].

Concerning the second direction, the objective is to find another means for congestion detection than losses. This can be done either by sending an explicit signal from network routers to TCP sources as with the Explicit Congestion Notification (ECN) proposal [61, 112], or by tracking the variation in round-trip time as with the Vegas proposal [45]. The objective of these proposals is to anticipate the congestion of the network and to react before the overflow of buffers. If all the sources, receivers and routers are compliant (according to Vegas or ECN), this will reduce drastically the number of congestion losses. The remaining losses could be then considered as mostly due to non-congestion. Moreover, if some averaging on the queue length is accomplished while detecting the congestion as with RED [65], transient overflow of buffers will not be detected by the sources as congestion events and hence the losses that result will be considered as not caused by congestion. Given that non-congestion losses only require retransmission without window reduction, the disappearance of congestion losses will lead to the definition at the source of a new TCP congestion control that reacts less seriously to losses. The congestion window can be reduced slightly or even not reduced upon the occurrence of a loss. However, it must be reduced by a factor of two in response to explicit congestion signals. This would provide an end-to-end solution to the problem of non-congestion losses.

Unfortunately, this ideal behavior does not exist in today's networks. In the absence of any explicit feedback from the network as with Vegas, congestion detection mechanisms at the sources may fail and here congestion losses are unavoidable. If the sources base their congestion control on an explicit information from the network as with ECN, some non-compliant routers will not provide the sources with the required information and drop packets upon congestion instead. A reduction of the window upon loss detection is necessary in these cases. For these reasons, end-to-end approaches still consider losses as congestion signals and reduce their windows respectively.

2.5 TCP and bandwidth asymmetry

The operation of TCP is strongly dependent on how well and how fast ACKs reach the source. Any loss or delay of ACKs will impact the window increase and the smoothness of the trans-

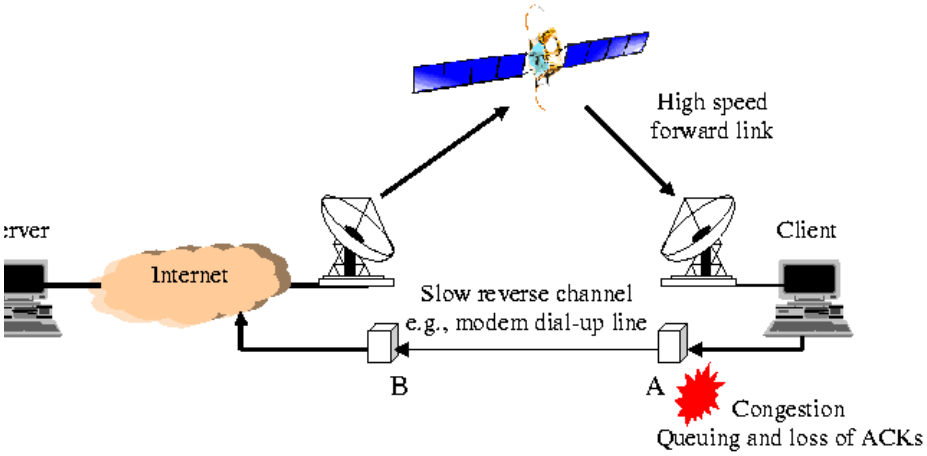


Figure 2.7: Example of an asymmetric path

mission. The reverse path from the destination to the source must be fast enough to carry the relatively high rate of ACKs generated by a TCP receiver. However, in today's Internet, many paths are presenting some bandwidth asymmetry between the two directions: high speed links (e.g., satellite down-links or cables) are used to download data from the Internet, and low speed channels (e.g., dial-up modem lines) are used to carry ACKs back to the sources (Figure 2.7). As examples of such paths we see the Direct Broadcast Satellite networks and Asymmetric Digital Subscriber Loop (ADSL) networks. Even if ACKs are of small size with respect to data packets, the slow reverse channel might be unable to carry the high rate of ACKs. This results in a congestion and losses of ACKs at its input (at point A in Figure 2.7) [3, 26]. If the reverse channel is also used for data, the problem of asymmetry will be exacerbated. Given the large size of data packets, this will also result in a problem of ACK compression [128]. Moreover, the contention between many connections for the small bandwidth of the reverse channel results in an unfairness between the different flows [85].

Before studying these points and the proposed solutions, let us define the asymmetry factor K. In [85], K is defined as the ratio of the bottleneck rate in the forward direction in terms of data packets per unit of time and the bottleneck rate in the reverse direction in terms of ACK packets per the same unit of time. This factor represents the traffic load on the reverse channel. A path is presenting some bandwidth asymmetry if

$$K = \frac{\text{Forward bandwidth}}{\text{Reverse bandwidth}} \times \frac{\text{ACK packet size}}{\text{Data packet size}} > 1.$$

First, we study the case of a single connection using the slow reverse channel for its ACKs. The impact of the congestion at the input of the slow channel is explained together with the proposed solutions. Second, we study the problems that appear when multiple connections contend for the reverse channel. The interaction between forward connections as well as between forward and reverse connections is highlighted.

2.5.1 Case of a single connection

The high rate of ACKs causes a queue building in the buffer at the input of the reverse channel (A in Figure 2.7). This increases the round-trip time of the connection and causes the loss of ACKs. The increase in the round-trip time results in a throughput deterioration [31] given that a TCP source is authorized to only transmit one window of packets every round-trip time. Also, it slows the growth of the window which impairs further the performance when operating on a long delay path or in a lossy environment.

Now, the loss of ACKs disturbs one of the main functionalities of the ACK stream, that of smoothing the transmission of TCP packets. When an ACK covering multiple packets arrives at the source, the window slides quickly and a burst of packets is transmitted. This may overwhelm the routers in the forward direction and deteriorate the performance [25, 85]. If the receiver acknowledges every data packet, it has been shown in [85] that one ACK over K is correctly received at the source. K is the asymmetry factor already defined. The result is the transmission of packets in bursts of K packets. The connection is supposed to be in the congestion avoidance mode. Thus, the buffers in forward routers must be at least of the size of the asymmetry factor [85]. Also, the loss of ACKs slows down the growth of the congestion window which results in a poor performance in case of long delay paths and lossy links. It has been shown in [85] that the throughput of a TCP connection is inversely proportional to the square root of K.

The proposed solutions to the problem of bandwidth asymmetry can be divided into two categories according to which side of the slow channel they are applied. Receiver side solutions try to solve the problem by reducing the congestion at the input of the reverse channel. The first solution, called header compression (e.g., SLIP [76]), compresses the header of ACKs on the slow channel (A-B in Figure 2.7) so that to increase its capacity in terms of ACK packets per unit of time. This solution profits from the fact that most of the informations carried by ACKs don't change during the connection lifetime.

The other solutions propose to reduce the rate of ACKs so that to avoid the congestion [25]. This alleviates the problem of the increase in RTT. Also, as we will see later, this solves the problem of fairness when many connections contend for the reverse channel. However, the loss of ACKs still exist (since the rate of the slow channel is not increased) and some solutions at the sender are required.

The first proposition to alleviate the congestion is to delay ACKs at the destination [25]. An ACK is sent every d packets and an adaptive mechanism has been proposed to change d as a function of the congestion on the reverse path. Another proposition [25] keeps the destination unchanged and solves the problem in the buffer at the input of the reverse channel (A). This solution profits from the fact that the information carried by ACKs is cumulative. When an ACK arrives at the slow channel (A), the buffer is scanned to see if another ACK from the same connection is queued. The old ACK is substituted by the new one. The queue is then maintained

at a small length which reduces the congestion and improves the performance. This solution is called *ACK filtering* since the flow of ACKs is filtered to match the rate of the reverse channel. The Chapter 8 of this thesis is devoted to this latter solution. ACK filtering looks as the most promising for improving the performance of TCP congestion control in asymmetric networks without the need to change the TCP protocol. We present first some problems with the scheme studied in the literature. We then propose some adaptive ACK filtering schemes together with an analytical model for optimization and performance evaluation.

Now, a solution at the source to the problem of burstiness caused by the loss of ACKs is not possible without any kind of packet spacing. The burstiness can be alleviated by imposing a limit on the size of bursts. But, in this case when ACKs are systematically lost, limiting the size of bursts will limit the throughput of the connection. Another solution has been proposed [25] to reconstruct the lost ACKs at the output of the slow reverse channel (B). When an ACK leaves the slow channel, all the missing ACKs between it and the previously received ACKs are generated. The generated ACKs are paced at a rate calculated from the average rate at which ACKs leave the reverse channel. This reconstruction may consist a solution to the problem of burstiness in asymmetric networks. However, the general problem of TCP burstiness upon ACK loss still exists.

2.5.2 Case of multiple connections

Consider first the case when multiple forward connections contend for the reverse channel (A-B in Figure 2.7). In the absence of any ACK filtering, the firstly starting connections will fill the buffer at the input of the slow channel (A). When a new connection arrives, it is very probable that its first ACKs get lost. The new connection then suffers from a slow window increase and possibly from timeouts. This results in a blockage of the new connection until the dominant connections end or reduce their rates [85]. The main cause for this blockage is that a drop-tail buffer treats all packets in the same manner. ACKs of a new connection, or those of a connection transmitting at a small window, must be given a certain priority until the congestion window reaches a large value. Once a large window is reached, the effect of ACK losses becomes less important. Such isolation requires an active buffer management technique (e.g., Flow RED [87]) that ensures fairness between the different flows of ACKs. We will address the problem of fairness between ACK flows with some details in Chapter 8.

Now, a forward connection can contend for the slow channel with a connection sending data in the reverse direction. But, the transmission time of a data packet on the reverse link is much longer than that of an ACK. This will decrease considerably the bandwidth available to ACKs which increases the asymmetry factor K. Also, ACKs are delayed after data packets which increases the round-trip time and the ACK loss rate. Now, if the forward connection is transmitting at a high rate so that its ACKs fill the buffer at the input of the slow channel, the data packets of the reverse connection will be frequently lost. The reverse connection then

reduces its rate whereas the forward connection continues increasing it since TCP congestion control does not react to the loss of ACKs. Again, this will result in a blockage of the reverse connection until the forward one ends [85]. An intelligent drop policy is required to prohibit a connection from occupying all the buffer. Now, to prohibit data packets from blocking for long time the reverse path, an intelligent scheduling policy can be envisaged at the input of the slow channel. The aim must be to serve packets according to their sizes. Many ACKs could then be served between data packets.

Another phenomenon noticed when many connections exist in the two directions is the *compression of ACKs* [128]. When ACKs arrive to a buffer after a data packet, they will be queued until the data packet is transmitted. The ACKs then reach the source in batches which results in an increase in burstiness and possibly in performance deterioration.

2.6 Conclusions

In this chapter we presented an overview of the different mechanisms of TCP congestion control. We summarized the different problems of the protocol that prohibit it from scaling to a large and heterogenous Internet. The different works in this wide area of research are outlined. We grouped the different works as a function of the network characteristic they deal with. We profited from this study to introduce the problems we will address in Chapters 7, 8, and 9. We also highlighted the importance of an analysis of TCP performance for the evolution of the Internet. We start this analysis in the next chapter by the end-to-end approach. We address the issue of end-to-end modeling of TCP and we seek for fine explicit expressions of the throughput. Before that, we detail on the benefits of these explicit expressions as well as on the different issues to be considered for a correct modeling of TCP congestion control. In the following three chapters (7, 8 and 9), we focus on the performance evaluation of the protocol in three challenging environments. Recall that the main tool we will use in our analysis is the modeling tool. Simulation and measurements will be used for validation of results and proposals.

Chapter 3

End-to-end modeling of TCP congestion control

The modeling of TCP congestion control is an important task for improving the service provided to Internet users and the efficiency of network resource utilization. A TCP source controls the rate of application packets as a function of the way the network treats or reacts to these packets. The main objective of modeling is to come up with simple explicit expressions of TCP throughput for a certain network reaction. This has two main advantages. First, it determines the factors impacting the performance of the protocol which gives people working on TCP, insights on how the congestion control has to be improved. For example, the modeling has shown that in case of drop-tail buffers and synchronized flows, the throughput of a TCP connection is inversely proportional to the square of the average round-trip time [84]. This has given an explanation to and an evaluation of the bias of TCP against connections with long round-trip times. The modeling has also shown that dropping packets randomly in network routers as with active queues (e.g., RED [65]), improves the fairness by making the throughput inversely proportional to the average round-trip time [12, 84, 92, 105]. Another possible use of an explicit expression of TCP throughput could be the study of a change of the parameters of TCP congestion control so that to minimize the variation of the window without adding to the aggressiveness of the protocol. This will be useful for multimedia applications using TCP, or possibly a new version of TCP adapted to such applications.

The second advantage of TCP modeling is that it permits network designers to improve the reaction of their network to incoming packets at the moment of congestion, given the current control policy of TCP. Important work has been done in this direction (e.g., [48, 58, 95, 97, 98]). The focus was and is always on the dimensioning of buffers in network routers and on the management of their occupancy. A typical problem in this direction is the tuning of RED (Random Early Detection) parameters [48, 58, 97]. For example, using an explicit expression of TCP throughput, it has been shown in [58] that when the number of connections exceeds a certain threshold, the RED buffer gets into an unstable regime. This instability has been explained by the sudden jump in the drop probability from p_{max} to 1 (see [65] for details on

RED parameters) when the average queue length exceeds the maximum threshold. This has motivated the introduction of a new parameter to RED (the *gentle_* parameter) in order to avoid such a jump. Other works have used the explicit expressions of TCP throughput to improve other parts of the network as the link layer on a wireless interface [32, 49]. Our work in Chapter 9 is a typical example of such works where we try to optimize the amount of FEC to be added to a bad quality link [32].

Recently, a new application of TCP modeling has emerged [62, 63]. It consists in using the explicit expressions of TCP throughput to control the rate of real time flows (e.g., audio flows) in a TCP-friendly way. The reaction of the network (e.g., packet drop probability) is averaged over a certain time interval and the rate of the real time flow is set to the expected throughput of a TCP connection running in the same network conditions. Clearly, a tradeoff exists between choosing a long averaging interval to get a smooth variation of the rate (hence a low jitter and a good quality at the receiver), and choosing a short interval to get a fast reaction to changes in network conditions.

The end-to-end modeling of TCP can be seen as a two-step procedure. First, we find the modeling of the evolution of TCP window between congestion events as well as upon congestion. This includes the modeling of some particular mechanisms of TCP such as the timeout and the limitation of the rate due to the window advertised by the receiver. Second comes the modeling of network reaction which consists, directly or after some transformation, in a modeling of the process of congestion events. The connection lifetime can be seen to be a succession of congestion events between which TCP increases its window and upon which it reduces it. As we will see later, a model for the network also has to describe how the round-trip time of the connection varies. The variation of the round-trip time determines the variation of the window and rate between congestion events. Note that TCP increases its window by the same amount during a round-trip time regardless of its length. An example of congestion events and window variation is depicted in Figure 2.2.

If the network is well defined (e.g., a single RED buffer), the modeling of TCP can be achieved with a considerable degree of correctness. One can come up with an accurate model for the network, combine it with a model for TCP, and solve the overall model for TCP performance. All the works assuming a single bottleneck router crossed by only TCP connections are of this type (e.g. [58, 95]). The difficulty exists when we want to approximate the throughput of TCP in the real Internet. Some assumptions are required in this case to model the reaction of this huge and heterogeneous environment. The difficulty of the analysis as well as the accuracy of the modeling change with the assumptions we make. For a given assumption, we must expect our results to hold on some Internet paths while not on others. The results may hold due to the correctness of our modeling on these particular paths. They may also hold due to another phenomenon that we observed during our work in this direction, that is the cancellation of errors introduced by the different blocks of a model for TCP. A good model for TCP must have all its

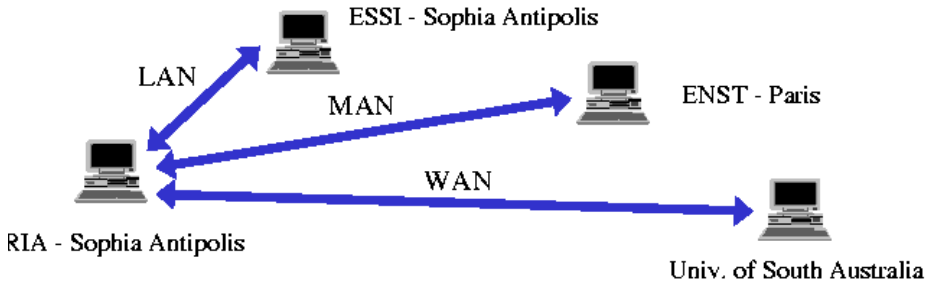


Figure 3.1: Measurement testbed

blocks validated separately and these blocks must work well on a wide range of Internet paths.

We address in this chapter the problem of TCP modeling in its general form. We focus mainly on the modeling of TCP in a real network rather than a particular environment as a single drop-tail buffer or a RED buffer. Based on measurements we conducted over the Internet, we discuss the different issues to be considered for a correct modeling of TCP. We outline the different approaches in this domain and we show the need for a general approach that copes with the heterogeneity of network reaction we observed. Along the discussion, we define our model for TCP window evolution as well as two models for the network: a simple Markovian model and a general model. We solve the two models for the explicit expression of the throughput in the following two chapters. At the end of this chapter, we introduce the technique we will use for the validation of our results. It consists in a separate validation of each block of a model for TCP instead of an overall model validation. But before starting, we present briefly the testbed we used for our measurements over the Internet. These measurements are only used in the first part of these thesis which deals with end-to-end performance evaluation of TCP. The results in the network-specific part are validated with simulations.

3.1 Measurement testbed

Our testbed consists of three long-life TCP connections run between a machine at INRIA Sophia Antipolis in the south of France and three other machines over the Internet (Figure 3.1). The first machine is located at ESSI next to INRIA. The second machine is located in Paris at about 800 Km from INRIA. The third machine is located at the University of South Australia. These can be considered as respectively LAN, MAN and WAN environments. In the sequel, we use the terms LAN, MAN and WAN to distinguish between the three connections. The source is located at INRIA and runs the New-Reno version of TCP [56, 64]. Recall that New-Reno is a recent version able to recover from multiple packet losses without timeout and with only one division of the window by two. The source is fed by a simple application that we wrote and that always has data to send. The connections were run for many hours in different days during the month of January 2000. We developed and ran a tool at INRIA that looks at the trace of every connection

(packets and ACKs) and calculates a certain number of statistics on the connection such as the total number of packets acknowledged, the total number of retransmissions, the moments at which the window is reduced, the variation of window and round-trip time over time, etc. We accounted for all the mechanisms of New-Reno when developing our tool, particularly the fact that a New-Reno source can recover from multiple packet losses in the same window of data. We stored the statistics at fixed intervals in separate files. We chose these intervals in a way to get enough data in each file. We shall assume that the network conditions are approximately the same during each interval. These intervals are set to 20 minutes for LAN, 40 minutes for MAN, and 60 minutes for WAN.

3.2 Modeling TCP window evolution

Most of the effort on TCP modeling [84, 92, 96, 97, 104, 105] has been devoted to long transfers, namely to the congestion avoidance mode of TCP. The slow start mode, due to its fast window increase and its short duration, has been often ignored. The small effect of slow start on long life connections compared to the effect of congestion avoidance can be seen in Figure 2.2. The window is assumed to increase linearly in time between congestion events and to decrease to half its size upon congestion. The moments of congestion are determined by the underlying model for the network. These are the moments at which the source detects the loss of a packet and decides to reduce its window. Ideally, and this is what the new versions of TCP try to approximate [56, 90], these must be the moments of detection of the first loss in a window of data packets.

TCP is known to increase its window in congestion avoidance mode in a linear manner but as a function of round-trip number rather than time (see Chapter 2). This increase is approximately equal to $1/b$ packets per round-trip time, where b is the number of packets covered by an ACK. For the time linear increase to hold, the round-trip is supposed to be constant or vary independently of the window, so it is substituted in the analysis by its average during the transfer [84, 92, 105].

For the multiplicative decrease factor, it is indeed equal to one half when Fast Recovery succeeds. If it is not the case, a timeout occurs, the window is set to one packet, and slow start is called to reach quickly the slow start threshold [121]. Some authors [105] keep the decrease factor in the case of timeout equal to one half while others [96] set the window to one packet and assume a linear increase from this low value. We believe that there will be no difference between the two approaches in the future given the expected low probability of timeout with the new enhancements proposed to TCP congestion control [4]. Our measurements showed that on WAN, most of the losses are detected with timeout and this is due to the small size of TCP window. The same result is noticed in [105]. On LAN and MAN, the window is large and the timeout phenomenon is quite absent.

In our work, we follow the same direction as the other works [84, 92, 96, 104, 105] concerning

the long duration of the TCP connection and the linearity of the window increase (or rate increase) between congestion events. We assume that the round-trip time is independent of the window size and we use its average in the analysis. Denote this average by RTT. Hence, the congestion window, that we denote by W, increases linearly with time at a rate $1/(bRTT)$. The rate of the TCP connection, which we denote by X and which is equal to the window size divided by RTT, also increases linearly with time at a rate $1/(bRTT^2)$. Concerning the multiplicative decrease factor, we take it equal to a constant value ν ($0 < \nu < 1$) for both kind of congestion detection methods (timeout and duplicate ACKs). Typically, ν is equal to 0.5. We will try in our analysis to avoid as much as possible the particular values of model parameters. This will make our results applicable to other congestion control policies of TCP.

Clearly, the above model is simple since it only accounts for the linear-increase multiplicative-decrease part of TCP congestion control. Other issues have to be considered for a better modeling of TCP. We will enumerate the most important ones in the following sections and introduce them into our model when possible.

3.2.1 Dependency between window and round-trip time

On paths where the window of the TCP connection is small compared to the bandwidth-delay product and where packets cross multiple congested routers, one should expect that the assumption on the linearity of the window increase will hold. Indeed, on such paths the connection does not contribute to the queueing time in network routers and the congestion is mostly caused by other aggressive connections. The round-trip time then varies independently of the window size and can be substituted in the analysis by its average [105] resulting in a time linear growth of the window (and hence of the rate) at a rate $1/(bRTT)$. However, this independency is not expected to hold on paths where the window of the connection is large compared to the bandwidth-delay product. The increase in the window in this latter case will result in a increase in the round-trip time which in turn will result in a sub-linear increase in the window (hence in the rate) with time [18]. Here, the overflow of network buffers will be mainly caused by the aggressiveness of the connection we are looking at. One should expect that, in presence of a linear model for TCP window evolution that uses the average round-trip time for the calculation of the window slope, this sub-linearity will result in an overestimation of the real throughput.

To understand such dependency, we plot the variation of the round-trip time as a function of the congestion window on our WAN and LAN connections (Figures 3.2 and 3.3). The WAN connection is a typical example of the first connection whereas the LAN connection is a typical example of the second one. We measure the round-trip time for one packet per window of packets and we note the window size upon this measurement. We then average the round-trip times obtained for close window sizes. This gives the thick line in both figures. We see clearly how on WAN, the round-trip time is on average constant and independent of the window. However, it is an increasing function of the window on LAN and a sub-linear window increase should be seen

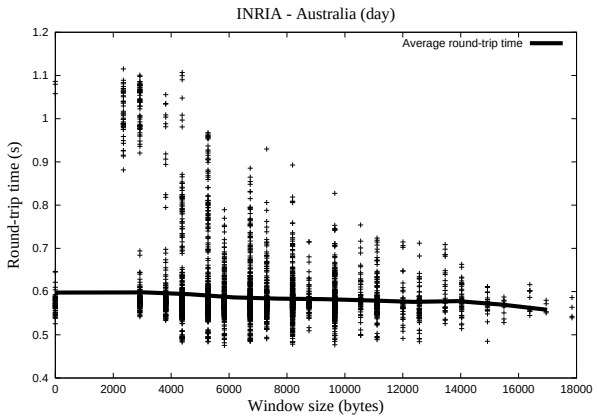


Figure 3.2: WAN: RTT vs. window

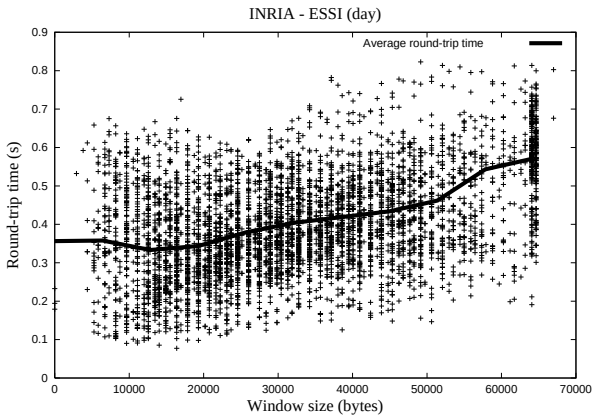


Figure 3.3: LAN: RTT vs. window

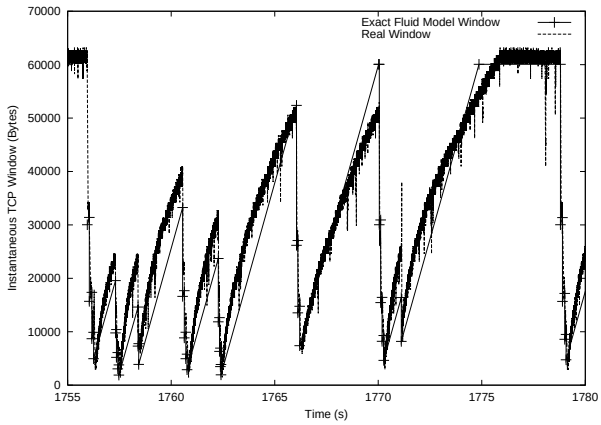


Figure 3.4: LAN: window vs. time

on this connection. Indeed, Figure 3.4 shows well this sub-linear behavior. We plot in this latter figure the variation of the real window on LAN for some seconds. The straight line corresponds to the expected window evolution if the round-trip time were independent of the window size. This line is given by our time linear model for TCP window evolution.

The problem with the sub-linearity of window increase is that it makes the analysis quite complicated and may be intractable within the general framework that we shall use. Moreover, it is not clear for the moment how it can be modeled. Its importance depends on many factors as the intensity of exogenous traffic sharing the path with the TCP connection, the buffering capacity in network routers, and the available bandwidth on the path. For these reasons, we skip this issue in this thesis and we keep it for our future research.

3.2.2 Modeling timeouts and Fast Recovery

The above model does not consider these two mechanisms of TCP [75, 121]. Due to the coarse granularity of TCP timers (500 ms in most implementations), the first mechanism introduces a certain idle time between a congestion and its detection. The second mechanism also introduces a certain time between the detection of a congestion with duplicate ACKs and the resumption of the window increase. During this latter time, the source is supposed to recover from losses and to transmit new packets in order for the ACK clock not to stop. The number of packets in the network during Fast Recovery has to be equal to the slow start threshold.

The modeling of these two mechanisms of TCP require a detailed description of the protocol behavior at the packet level. This behavior is quite complicated to model and varies from one version to another (see Chapter 2). The Fast Recovery phase has been always ignored due to the complexity of its modeling, also because it adds a negligible contribution to the throughput when it works well. It has been assumed in [105] that once the source receives three duplicate ACKs, the window resumes its increase until the next congestion event. This seems reasonable with the new versions of TCP (e.g., SACK [56]) where the Fast Recovery phase is quite robust and fast, but it is not true with the other versions as Reno where Fast Recovery may fail due to multiple losses per window. A failure of Fast Recovery results in a timeout and a slow start from a window of one packet. A model which does not account for the Fast Recovery phase will lead to throughput overestimation if such failures are frequent. Given the complexity and the continuous evolution of this phase, we skip it from our model.

The timeout mechanism is studied in [105] using the probability that a packet is dropped. Denote this probability by p. The authors focus on the calculation of the probability that the source fails to receive three duplicate ACKs to trigger Fast Recovery. This has been assumed to be the necessary and sufficient condition for the occurrence of a timeout. Timeouts are assumed not to occur during the Fast Recovery phase. We believe that this will be the case with the new versions of TCP that use the SACK option [91] and that have a robust and a quick Fast Recovery phase. We also believe that with the modifications proposed to TCP error recovery in [4], the failure to receive three duplicate ACKs will no longer be a sufficient condition to get a timeout. Sources will be able to recover from losses when more than one ACK are received per round-trip time. In the wait for these modifications to be deployed, we will show in the following a heuristic for the addition of the timeout mechanism to our model according to the condition in [105]. The advantage of our heuristic compared to that used in [105] is that can be used in the general framework we shall consider.

To simplify the analysis, it has been assumed in [105] that when a packet is dropped, the subsequent packets in the same window are also dropped. The probability to get a timeout ($Q(p)$) as well as the average duration of a timeout period ($Z(p)$) are calculated as a function of

p,

$$\begin{aligned} Q(p) &= \min\left(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}\right), \\ Z(p) &= T_0 \frac{1+p+2p^2+4p^3+8p^4+16p^5+32p^6}{1-p}, \end{aligned}$$

with

$$w = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2},$$

and T_0 the basic timeout interval which is doubled when the source backs off its timer after the loss of a retransmission. As we saw in Section 2.1.3, T_0 is set by TCP to $SRTT + 4 * RTTVAR$. In practice, 4RTT has been shown to be a safe value for T_0 [63].

Using these two functions of p, the impact of timeout intervals can be introduced into our model in the following way. We calculate first the throughput of the connection when excluding these intervals, in other words when assuming that the window resumes its linear increase directly after a congestion event whatever is the method of detection (Figure 3.5). Denote this throughput by $\bar{X}_d$. Denote by $\bar{X}_f$ the throughput of the connection in the presence of timeout intervals (Figure 3.6). As we will see in the next chapter, the throughput in both cases is equal to the ratio of the average number of packets that cross the network between two congestion events ($\simeq 1/p$) and the average time between congestion events. Let $\{S_n\}$ denote the process of times between congestion events when timeout intervals are excluded (Figure 3.5). Let $\{S'_n\}$ denote the process of times between congestion events when timeout intervals are considered (Figure 3.6). Given that the average number of packets that cross the network between two congestion events is the same in both cases ($\simeq 1/p$), it follows that

$$\bar{X}_f = \bar{X}_d \frac{\mathrm{E}[S_n]}{\mathrm{E}[S'_n]},$$

and

$$\bar{X}_d = \frac{1/p}{\mathrm{E}[S_n]}.$$

Using these two equations and the fact that

$$\mathrm{E}\left[S'_n\right] = \mathrm{E}[S_n] + Q(p)Z(p),$$

we can write

$$\bar{X}_f = \frac{\bar{X}_d}{1 + p\bar{X}_d Q(p) Z(p)}. \tag{3.1}$$

Note that our expressions for the throughput are in terms of packets/s. Note also that our throughput corresponds to the rate at which TCP packets leave the network, also called the receiving rate.

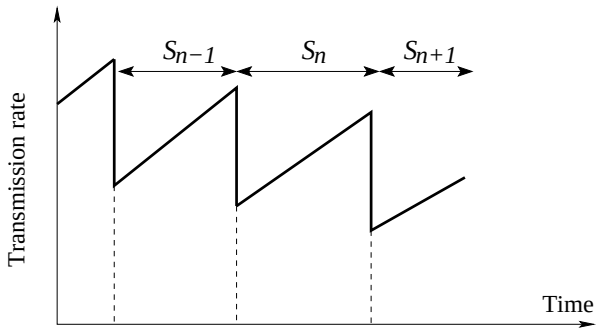


Figure 3.5: Model without timeout intervals

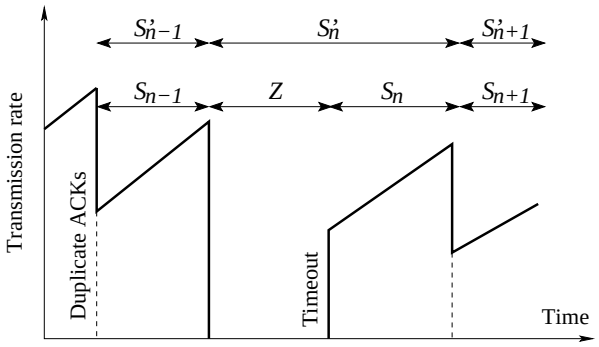


Figure 3.6: Model with timeout intervals

3.2.3 Modeling window limitation

When the congestion window reaches the receiver window, the rate increase stops until the next congestion event. This phenomenon can be seen in Figure 3.4 where we plot the number of bytes in the network as a function of time. The model becomes sub-linear and the calculation of a simple expression for the throughput seems to be impossible except for some particular network reactions. For example, the calculation is straightforward when the time between congestion events is constant. In [96], the authors propose a model for congestion events that appear according to a Poisson process without finding an explicit expression for the throughput. In our work in this direction, we succeeded to solve the problem when congestion events appear not only according to a Poisson process, but also according to a batch Poisson process. We used for this purpose sophisticated techniques from M/G/1 queueing theory [80]. The complete analysis as well as the validation of results are presented in Chapter 6.

For more general models for the network (e.g., a generally distributed time between congestion events), one can always think about finding some approximations of the throughput. One possible approximation [105] is to assume that the receiver window is always reached between congestion events. The problem is then automatically transformed into a simpler one with constant times between congestion events equal to the average of these times. This is the kind of approximation we will use with our Markovian model for the network (Chapter 4). With our general model, we will proceed in a different way. We will find bounds for the throughput that are also a good approximation (Chapter 5). The advantage of the latter approximation is that it is valid for any receiver window. The first approximation is only valid for small receiver windows, so one needs to condition on the average window upon congestion in order to use either the approximation or the expression of the throughput obtained when there is no window limitation. We will explain this difference between the two approximations when we present our calculation of the bounds for the throughput in Chapter 5.

Note that the problem of window limitation exists because of the small windows advertised by current Internet receivers (less than 64 Kbytes). But, this is supposed to disappear in the

future with the trend to increase the window field in the TCP header and the trend to change dynamically the buffer size allocated to the TCP connection at the receiver [118]. It is even recommended in [5] that future studies on TCP congestion control should consider an infinite receiver window. Models that account for the limitation of the rate will not be of big importance.

3.2.4 Fluid models versus discrete models

Some of the models for TCP assume that the window increases continuously between congestion events [18, 92, 96, 97, 117]. The use of a continuous model for the window facilitates the analysis since it permits the use of tools from the theory of continuous functions as integration and differentiation. The continuous increase may hold for the congestion window at the source but it does not hold for the volume of data in the network. Indeed, the former quantity increases in small values (bytes) upon ACK arrivals, however the latter one increases in steps of one packet (generally 1460 bytes) every b round-trip times [105]. This is due to the Nagle algorithm [99] which prohibits the source from injecting small packets into the network. At the moment of congestion, it is the congestion window at the source which is divided by two rather than the number of packets in the network [121]. Thus, models assuming continuous increase of the window, also called *fluid* models, are appropriate for the prediction of the window variation at the TCP source. The throughput obtained with these models is the throughput that the TCP connection would realize if it is not limited by the Nagle algorithm.

The correct throughput can be obtained by one of two ways: either by using a discrete model for the window evolution as the one in [105], or by using a fluid model and introducing some corrections into the calculated throughput so as to account for the difference between the congestion window at the source and the volume of data in the network. In our work, we choose the second way for the calculation of the throughput given the simplicity of the analysis the use of fluid models permits. We first calculate the throughput when the window increases continuously as a function of time between congestion events. We then subtract from this throughput, the rate of packets that would have crossed the network if the Nagle algorithm did not exist. We also subtract the rate of packets that are lost in the network upon congestion. Recall that we look at the throughput as the average rate of packets that leave the network.

To illustrate the corrections we introduce into the throughput given by the fluid model, we plot in Figure 3.7 the number of packets that cross the network between two congestion events for both a fluid model and a discrete (or a packet) model. The line for the discrete model is taken from [105]. On average, half of the window is assumed to be dropped in the network upon congestion (the last round-trip time in the figure). Suppose that all the rates are expressed in terms of packets/s. Denote by $\mathrm{E}[W_n^*]$ the average window size upon congestion and by $\mathrm{E}[S_n]$ the average time between congestion events. Let $\bar{X}$ denote the throughput obtained with the fluid model and $\bar{X}_d$ the real throughput of TCP. A good approximation of $\bar{X}_d$ can be obtained by:

- Shifting down the fluid window by 0.5 packet which results in a decrease in the throughput $\bar{X}$ by $0.5/RTT$.
- Subtracting the rate of dropped packets which is approximately equal to

$$\text{Rate of dropped packets } = \frac{1}{2}\frac{\mathrm{E}\left[W_n^*\right]}{\mathrm{E}\left[S_n\right]}.$$

It follows that

$$\bar{X}_d = \bar{X} - \frac{0.5}{RTT} - \frac{0.5\mathrm{E}\left[W_n^*\right]}{\mathrm{E}\left[S_n\right]}.$$

Using one of our results in Chapter 5 (Equation (5.3)) which says that

$$\mathrm{E}\left[W_n^*\right] = \frac{\mathrm{E}\left[S_n\right]}{bRTT(1-\nu)},$$

we can write

$$\bar{X}_d = \bar{X} - \frac{0.5}{RTT}\left(1 + \frac{1}{b(1-\nu)}\right). \tag{3.2}$$

Finally, the throughput $\bar{X}_d$ has to be plugged into Equation (3.1) in order to account for timeouts. With this correction and as we will see later when validating our model, a fluid model is able to give the same throughput as a detailed discrete model, of course if both of them model the network in the same way.

From now on, we will only focus on the calculation of the throughput of a fluid model without timeout intervals. The window (or the rate) increases continuously between congestion events, decreases with a factor ν upon congestion, and resumes directly its increase after that. In our analysis we will work with both rate and window. At any time, one can switch from the first to the second by multiplying the rate by RTT and from the second to the first by dividing the window by RTT. Once the throughput of this fluid model is calculated (that we denote by $\bar{X}$), we must plug it into Equations (3.2) and (3.1) in order to get the final result of our model that accounts for timeouts and the packet nature of TCP (that we denote by $\bar{X}_f$). It is this final result that must be used as an approximation of TCP throughput.

3.3 Modeling the network

This is the part of the modeling where the heterogeneity of the Internet has the greatest impact. The objective of this part is to find a good characterization of congestion moments or loss moments. A loss moment in our terminology is the moment at which the TCP source decides that the network is congested and that it must reduce its window. These moments can be directly characterized by making some assumptions on the way they appear. For example, one can assume that they appear according to a deterministic process or a Poisson process. This direct characterization is the approach widely used in the literature [92, 96, 105]. Another possible but indirect characterization consists in finding a model for the reaction of the network

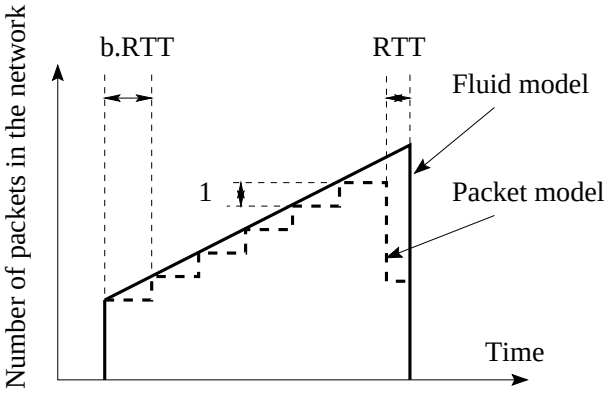


Figure 3.7: Fluid model vs. discrete model

to a particular TCP packet or during a small time interval. As an example of this indirect approach we find the works that suppose that TCP packets are dropped within the network with a constant or variable probability [95, 104], or those assuming that loss moments form a Poisson process with a variable intensity function of the window size of the connection [117]. The advantage of the indirect approach is that it decouples the model for the network from the control policy at the TCP source. It happens that on some Internet paths, the process of losses seen by a TCP connection is a function of the way it increases and decreases its window, and that this process changes if another control policy is used, for example if packets are transmitted at a constant rate. The indirect approach is very useful on such paths since it permits to deduce the loss process that a TCP connection will see from the loss process seen by another connection with a different congestion control policy. We cite different applications of the indirect approach on such paths. The first application is that one can probe the network with a certain flow of packets (e.g., a constant rate flow) and calculate the parameters of an appropriate model for the network (e.g., packet drop probability, variation of loss intensity as a function of transmission rate). With these parameters, it will be possible to predict the performance of a TCP connection on the same path. Another application of the indirect approach is that from the trace of a TCP connection, one can build a model for the network and predict the performance of another TCP connection with another congestion control policy. This will be useful for a study of the impact of a change of TCP congestion control parameters. A third application of the indirect approach is that a TCP-friendly application (e.g., TFRC [63]) can deduce the loss process that a TCP flow would see from the loss process it sees, and hence can get a better estimate of the rate to use. All these applications are not possible with the direct approach since, without a model for network reaction, we cannot deduce the performance of a TCP connection from the loss process seen by another connection with another control policy. Recall that we are talking about paths where the loss process changes with the congestion control policy.

The difficulty with the indirect approach is in the definition of a correct model for the Internet

and the calculation of its parameters. This may be possible for a simple network of one router (e.g., a RED buffer is known to drop packets with a probability that increases linearly with the average queue length [65]) but it seems to be very difficult for a wide network as the Internet. It is not clear if there exists a model for the Internet that works on all paths. Certainly, the Internet reaction to TCP packets changes from one path to another and along time on the same path. For example, on some paths the network may drop packets with a constant probability, on other paths with a probability that increases linearly with the congestion window, on other paths with a probability that increases logarithmically with the congestion window, etc. One can imagine different models for the network. The question is on how many paths these models are useful.

In the first part of this thesis, we shall only focus on the direct approach. Recall that the direct approach consists in using the parameters of the loss process seen by a TCP connection for throughput calculation. Our main objective is to show how much simplistic assumptions on the loss process as those made in the literature (e.g., deterministic [92, 105], Poisson [96]) impact the accuracy of the modeling and to come up with an explicit expression of the throughput that works with any distribution of times between loss moments. Once derived, this expression could be used to study how the throughput of TCP varies with the distribution of times between losses. For example, one of our important results is that the throughput of a TCP connection increases with the increase in the variance of times between losses or what we call in Chapter 4 the burstiness of losses [11, 12, 13]. On paths where the loss process does not change with the congestion control policy, the explicit expression of TCP throughput will tell us how the performance of TCP changes with any change in the parameters of TCP congestion control as the round-trip time, the window increase rate, the reduction factor, etc.

First, we start by presenting some measurement results to show how much the Internet is heterogenous and how much the loss process can change from one path to another. This has motivated us to adopt two different approaches for modeling the loss process: a Markovian approach and a general approach. The calculation of the throughput using these two approaches is presented in the next two chapters.

3.3.1 Diversity of loss processes in the Internet

We present in this section some of the loss processes we found on our three connections (Section 3.1). We first plot the distributions of inter-loss times. Figures 3.8, 3.10 and 3.12 show what we got respectively on the LAN, the MAN and the WAN connections. The three figures contain for comparison some theoretical distributions (Exponential for LAN and WAN, Normal on MAN). On LAN, the process is highly bursty which results in this pulse close to the origin. This burstiness can be also seen in Figure 3.4 where the window is divided multiple times by two in short time intervals. We noticed that the congestion on LAN stays for multiple consecutive round-trip times during which the network keeps dropping packets and the source keeps reducing

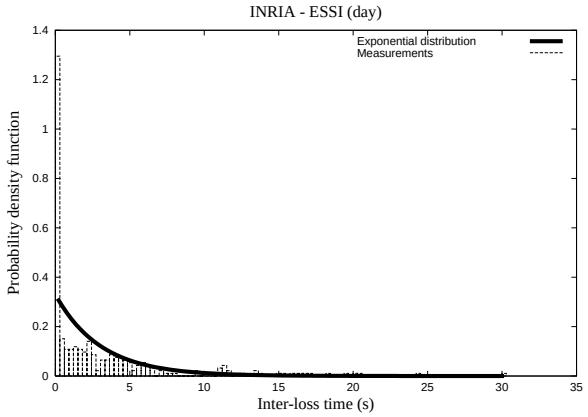


Figure 3.8: LAN: Inter-loss time distribution

Hour (Traces of 20 min)	Covariance coefficient $Cov(S_n, S_{n-1})/Var(S_n)$
11:00	+ 0.034
12:00	+ 0.041
12:30	+ 0.113
13:00	+ 0.001
13:30	- 0.191
14:00	- 0.078

Figure 3.9: LAN: Covariance coefficient

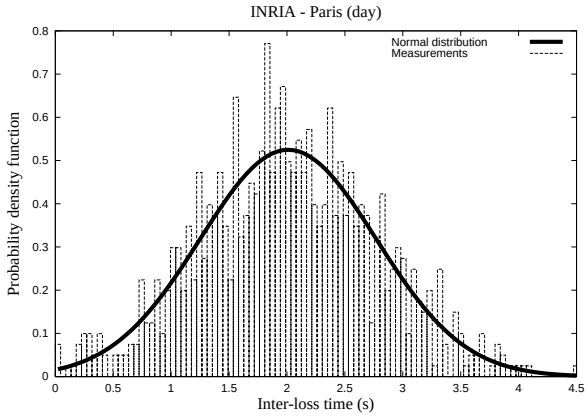


Figure 3.10: MAN: Inter-loss time distribution

Hour (Traces of 40 min)	Covariance coefficient $Cov(S_n, S_{n-1})/Var(S_n)$
15:00	+ 0.106
19:00	+ 0.101
20:00	+ 0.015
21:00	- 0.01
22:00	- 0.048
23:00	- 0.005

Figure 3.11: MAN: Covariance coefficient

its window. On MAN, the times between losses follow well a Normal distribution. On WAN and as one must expect, the loss process is close to Poisson. Indeed on WAN, the source has a small window and does not contribute to the congestion of the network. The loss process it sees is the superposition of a large number of processes in all the routers it crosses. We also found some correlation of losses on the three connections. This correlation varies during the day between negative and positive values with an absolute value of the covariance coefficient reaching sometimes 0.2. Tables 3.9, 3.11 and 3.13 show some of the covariance coefficients we saw on the three connections at different hours during the day. We define the covariance coefficient (of order 1) of a process $\{S_n\}$ as follows,

$$Cov = \frac{\mathrm{E}\left[S_n S_{n-1}\right] - \mathrm{E}\left[S_n\right]^2}{\mathrm{E}\left[S_n^2\right] - \mathrm{E}\left[S_n\right]^2}.$$

This coefficient varies between -1 when inter-loss times are highly negatively correlated and +1 when they are highly positively correlated.

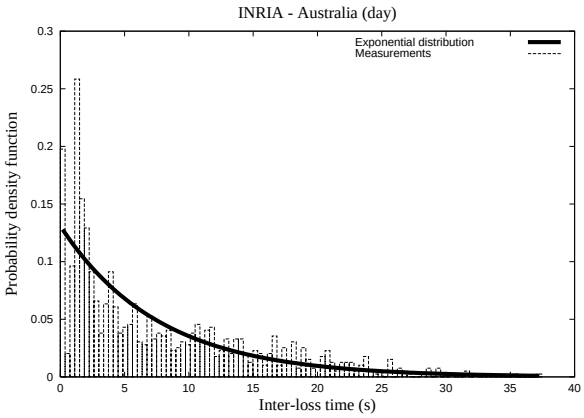


Figure 3.12: WAN: Inter-loss time distribution

Hour (Traces of 60 min)	Covariance coefficient $Cov(S_n, S_{n-1})/Var(S_n)$
11:00	- 0.197
12:00	- 0.001
14:00	- 0.102
16:00	- 0.107
20:00	+ 0.023
22:00	- 0.09

Figure 3.13: WAN: Covariance coefficient

The results we presented are some examples of losses that a TCP connection may see over the Internet. We expect to see other processes on other paths. Other distributions of inter-loss times could be found. Also, some paths as those including satellite and wireless links, may exhibit more memory than our paths which will result in a more important correlation of losses. All this means that simplistic assumptions on the loss process as those made in the literature (e.g., deterministic [84, 92, 105], Poisson [96]) are not enough for an accurate modeling of TCP. There is a need for a model for losses that can cover most if not all Internet paths. Also, there is a need for a study of the impact of the way with which losses occur on the throughput of the protocol. We define in the following section the models for losses that we will use in this thesis and that will permit us to achieve the above objectives. The corresponding analysis is presented in the following chapters.

3.3.2 Modeling the loss process

We use three models for the loss process in this thesis. They can be seen as three point processes where the occurrence of a point corresponds to the occurrence of a loss or a congestion event. The first two models are general and are related to the case when there is no limitation on the congestion window. The third model is less general and it is related to the case when such limitation exists. We briefly explain the three models in the rest of this section. The mathematical formulation of each model is presented before the analysis in the corresponding chapter.

Our first model for losses is based on a Markovian approach. It consists on a path changing its state according to a Markov chain. The loss rate (i.e., number of losses per unit of time) changes when the path switches from one state to another. We present in Chapter 4 the analysis for a simple version of this model using a two-state Markov chain to model the path. This analysis can be found in [11, 13]. The general version as well as the corresponding analysis can be found

in [14]. The transition of the Markov chain associated to the path as well as the occurrence of losses happen at some potential loss moments that we introduced. This is similar to what happens with a Markovian Arrival Process (MAP) [20]. With a particular case of this model we prove our first result, that the throughput of a TCP connection increases when losses become clustered. This increase can reach 100%. We also present a technique for the identification of the parameters of the loss process (e.g., transition matrix of the Markov Chain, rate of losses in each state) from the trace of a real TCP connection.

Our second model for losses is the most general one. It covers the different distributions and the different levels of correlation of inter-loss times. We work directly with the process $\{S_n\}$. The only assumption we make on this process is that it is stationary ergodic. We derive a simple expression of the throughput function of the intensity of losses, the variance of inter-loss times and all the covariance coefficients. It is a generalization of the well known square root formula [92, 105] for TCP throughput calculated under the assumption that the loss process is deterministic. Our expression can be easily specified to any loss process. It shows clearly the impact of the different parameters of the loss process on the performance. In particular, it shows that the throughput of TCP increases with the variance of times between losses and that the weights of covariance functions decrease geometrically. One can find in [12] our analysis and results for the case of a general model for losses.

In case of window limitation, the calculation of the throughput with the previous models for losses seems to be impossible given the complexity of the models. In chapter 6, we find the expression of the throughput when batches of losses appear according to a Poisson process. The number of losses within a batch may follow a general distribution. Our analysis also permits to find the distribution of the congestion window under this particular distribution of losses. We will see that, even though the loss process is not very complicated, the calculation of the throughput is quite hard. This calculation as well as the validation of our results can be found in [15, 16].

3.4 Separate model validation

We introduce in this section the notion of separate validation of each part of a model for TCP. This is the technique that we will use for the validation of our results in the following three chapters. Researchers compare directly the real throughput realized by a TCP connection to the final result of their modeling. But as we saw, a model for TCP is composed of two parts: a model for the window evolution and a model for losses. Proceeding for the validation in one step will hide the errors introduced by these two parts. This will give the sum of the two errors instead of each of them. First, this will preclude us from distinguishing from which part of the model the error is mostly due. Second, and this is the most important, the errors introduced by the two parts of the model may be of opposite signs which may make the total error small and

acceptable. The result will be a wrong estimation of the capacity of the model since, as we will see later, this phenomenon of error cancellation does not always exist.

To avoid the problem of error superposition and possibly error cancellation, we propose to validate separately the two parts of the model. We start first by the model for losses. Our fluid linear model is used for window evolution. To only get the error introduced by the assumption on the distribution of losses, the window of TCP should increase continuously and linearly between losses and decrease multiplicatively by a factor ν upon losses. But, TCP window does not have this ideal behavior in reality. What we do here is to construct this ideal behavior of the window using the moments of losses seen by the TCP connection. The average of round-trip time measurements is used for the calculation of the linear increase rate. The ideal window is shown by the straight line in Figure 3.4. We call the version of TCP that has the window behaving as the ideal window "ideal TCP" or "exact fluid model". Then, we calculate numerically the throughput obtained by ideal TCP and we compare it to the result of our modeling under a certain assumption on the loss process. The comparison gives us the error introduced by the model for losses.

The throughput of ideal TCP is calculated as follows. First, we sum over all the areas between the ideal window and time, then we divide this sum by the total time of the measurement. This gives the time average of the ideal window. The ideal throughput is obtained by dividing the average of the ideal window by the measured average round-trip time. Now, to get the error introduced by the model for the window evolution, all what we need to do is to compare the ideal throughput to the real throughput. Before this comparison, the ideal throughput has to be corrected for timeouts and the packet nature of TCP using for example the heuristics we gave in (3.1) and (3.2).

We present some results to confirm the utility of such a method for validation. We take first the LAN connection. We plot in Figure 3.14 the ideal throughput we obtained during the different hours of the day. We also plot in the same figure: the real throughput, the result of our model when assuming deterministic inter-loss times, and the result of our model when assuming that the loss process is Poisson. A model assuming deterministic losses underestimates the ideal throughput given that the high variance of inter-loss times we observed on LAN (Figure 3.8). The ideal throughput in turn overestimates the real throughput due to the sub-linear growth of the window we discussed in Section 3.2.1. We notice that a direct comparison of the real throughput to the result of the modeling in case of deterministic losses hides the two errors and gives us an impression that the model works correctly. Now, when assuming that losses form a Poisson process, we approximate better the ideal throughput but worse the real throughput. According to a direct validation technique, the overall model is becoming worse with the Poisson process which is a wrong conclusion. With our technique however, we are able to understand that the Poisson process is better than the deterministic process, and that the real problem is with the model for window variation. Hence, in this particular case, our technique reveals a

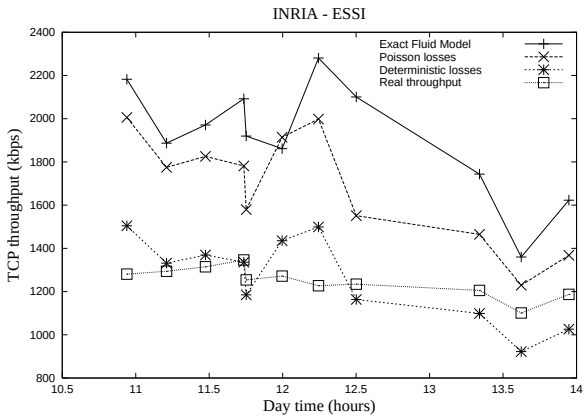


Figure 3.14: LAN: separate model validation

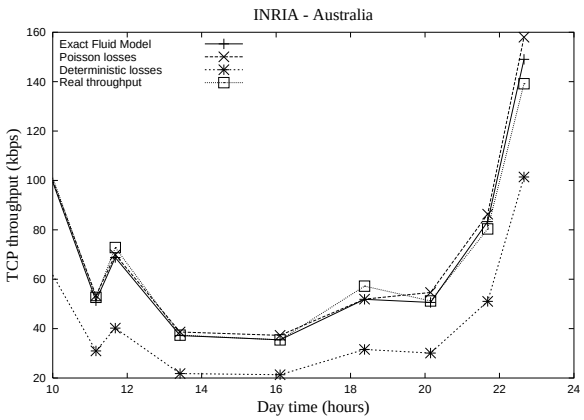


Figure 3.15: WAN: separate model validation

problem that was hidden.

We take now the WAN connection (Figure 3.15). The ideal throughput approximates well the real throughput given that the window increase on WAN is quite linear (Section 3.2.1). However, our model with deterministic losses does not give good performance in this case as on the LAN connection since the loss process is close to Poisson. Assuming that losses form a Poisson process gives better performance. This difference in the performance of a model with deterministic and Poisson losses between LAN and WAN cannot be explained without the method of validation we introduced. Due to our method, we conclude that it is better to take the loss process as Poisson on the WAN connection. On the LAN connection, it is better to take losses as Poisson but we have to correct the part of the model concerning the window variation.

3.5 Conclusions

We presented in this chapter an overview of the different issues to be considered when modeling TCP. We introduced our models that we will analyze in the following three chapters. Our main results in this chapter can be summarized as follows:

- A linear window increase model does not hold on paths where the round-trip time is dependent on the window size. A sub-linear window increase model needs to be used in this case. We keep the modeling of such sub-linearity for our future research. We only focus in this thesis on linear models for TCP window variation.
- The process of congestion events needs to be well characterized given the diversity of loss processes on Internet paths. This has been the main motivation behind the general models we introduced and that we are going to analyze.
- The validation of a model for TCP needs to be done in two steps: the model for window evolution and the model for the network need to be validated separately. A one-step

validation hides the errors introduced by these two models and makes the behavior of the overall model unexplainable in some situations.

Chapter 4

Modeling TCP congestion control: a Markovian approach

The first approach that we consider for modeling the network consists in a loss process that changes its rate according to a Markov chain. For simplicity of the presentation, we consider the case of a two-state Markov chain. The analysis for the case of a multi-state Markov chain can be found in [14]. The intensity of losses is no longer uniform as with previous models for TCP [60, 84, 92, 96, 105] but changes along time. The change in loss rate can be caused by the fluctuation of Internet traffic. It can be also caused by some particular transmission media that suffer from a fluctuation of their bit error rate. For example, it is well known [113] that due to the mobility of users and the superposition of reflected and non-reflected signals, the signal to noise ratio in a wireless network oscillates between low and high values causing a change in the loss rate. This is called the fading phenomenon and theoretical models (e.g., Rayleigh model [113]) have been proposed to quantify it. It is also known that losses on a satellite link appear in bursts [55]. Our first objective in this chapter is to evaluate the impact of such a fluctuation of loss rate on TCP throughput. In particular, we are interested in evaluating the impact of the burstiness (or the clustering) of loss events on the performance of TCP congestion control. Our second objective is to provide a tool for the prediction of TCP throughput on such non-homogeneous paths. We validate such a tool on our LAN connection (Section 3.1) where the loss process presents an important degree of burstiness. Note here that in contrast to other works in the literature that study the impact on TCP of bursts of losses at the packet level (i.e., all packets are assumed to be lost when the path is in the bad state) [49, 51, 83], our focus is on the burstiness of loss events or congestion events.

4.1 The model

Consider a linear-increase multiplicative-decrease fluid model for the rate evolution of a TCP connection. Denote by $X(t)$ the transmission rate at time t. It is approximately equal to the window size $W(t)$ divided by the average round-trip time RTT. The rate increases linearly at

a rate $\alpha = 1/(bRTT^2)$ between congestion events and decreases multiplicatively by a constant factor $\nu \in (0,1)$ upon congestion. b represents how many data packets are covered by an ACK. Typically, $b = 2$ [105]. The throughput of TCP that we intend to calculate is given by,

$$\bar{X} = \lim_{t \to +\infty} \frac{1}{t} \int_0^t X(\tau) \mathrm{d}\tau. \tag{4.1}$$

We call this entity the throughput even though it needs to be corrected with Equations (3.1) and (3.2) in order to account for timeouts and the packet nature of TCP. $\bar{X}$ is equal to the time-average of the transmission rate which in turn is equal to the expectation of the transmission rate at any moment ($\mathrm{E}[X(t)]$) if the process $X(t)$ is stationary ergodic [40].

Let us now formulate mathematically our model for the network. The path between the source and the destination is assumed to be in one of two states: *Good* and *Bad*. We let losses appear in both states of the path. The difference between the two states is that losses appear with different rates with a smaller rate in the Good state. This model for the network is slightly different than the famous two-state Gilbert model [66] often used in the literature for the study of wireless networks [32, 49, 51, 83] and where packets are only lost in the Bad state. We will return to the Gilbert model in Chapter 9 when we present our study of TCP performance in wireless networks. To model the change in the loss rate between the two states of the path, we define a series of potential losses. The decrease in rate and the change of path state are only possible at potential loss moments. The difference in loss rate between the two states of the path comes from a difference in the probability with which a potential loss is transformed into a real loss. This is similar to a Markovian Arrival Process (MAP) [20, 100] in which at each state transition an arrival (a loss event in our terminology) can occur with a probability that depends on the state.

Let t_n denote the time at which the nth potential loss occurs. Let $\{D_n\}, n = 1, 2, \ldots,$ be the sequence of times between potential losses: $D_n = t_{n+1} - t_n$. The D_n are assumed to be i.i.d. (independent and identically distributed) with expectation d, k th moment $d^{(k)}$ and Laplace Stieltjis Transform $D^*(s)$. Let X_n be the transmission rate of TCP just prior to the nth potential loss. As we said, potential losses are transformed into real losses with a certain probability function of the state of the path. Let Y_n be the state of the path at the moment of the nth potential loss. We consider the states B (for Bad) in which a potential loss is transformed into a real loss with probability p_B, and G (for Good) in which it is transformed with a probability p_G. We shall assume throughout that $p_G \leq p_B$ and that $p_B > 0$. We further assume that the sequences $\{Y_n\}$ and $\{D_n\}$ are independent. For explanation, we show in Figure 4.1 an example of the variation of TCP rate on such Markovian paths.

The state of the path $Y_n \in \{B, G\}$ is assumed to be a Markov chain with the following transition matrix (Figure 4.2),

$$P = \begin{bmatrix} \gamma & \bar{\gamma} \\ \bar{\beta} & \beta \end{bmatrix}.$$

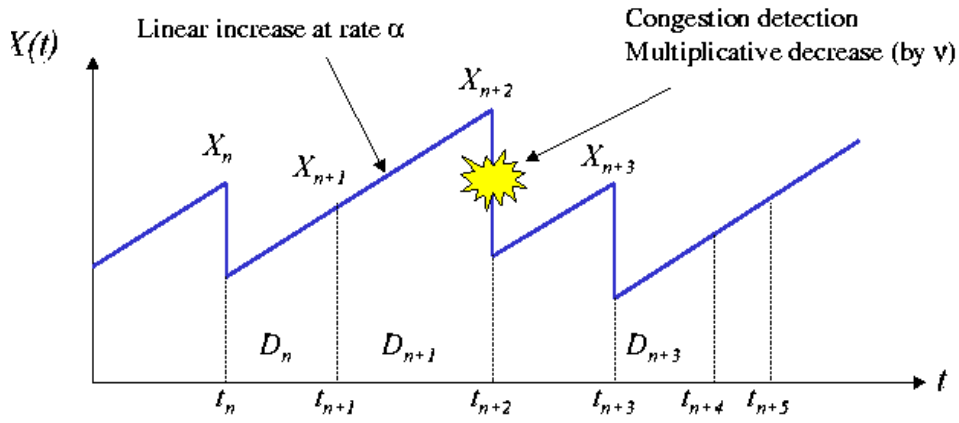


Figure 4.1: Variation of TCP rate on a Markovian path

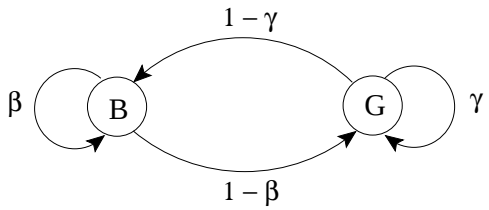


Figure 4.2: The Markov chain associated to the path

State 1 (resp. 2) corresponds to the Good (resp. Bad) state of the path. We shall assume throughout that $\gamma, \beta \in (0,1)$. $\{Y_n\}_{n=1}^{+\infty}$ is then ergodic with stationary probabilities,

$$\pi_G = \frac{1-\beta}{2-\beta-\gamma} = \frac{\bar{\beta}}{\bar{\beta}+\bar{\gamma}}, \qquad \pi_B = \frac{1-\gamma}{2-\beta-\gamma} = \frac{\bar{\gamma}}{\bar{\beta}+\bar{\gamma}}.$$

The average rate of losses over all the states is given by,

$$\lambda = \frac{p_G \pi_G + p_B \pi_B}{d},$$

with $\lambda_G = p_G/d$ and $\lambda_B = p_B/d$ the average rates of losses in the Good and Bad states respectively. With this formulation of the loss process, we are able to vary the average loss rate as well as the burstiness of losses. We will explain this later when we study the impact of the distribution of losses on TCP performance. Let us now calculate the throughput of TCP under the already described loss process.

4.2 Performance Analysis

Define the two random variables U_n and V_n describing the behavior of the transmission rate (reduction or no) when a potential loss occurs. They correspond to the two states of the path. A value one of these variables means that the potential loss causes a reduction of the transmission

rate. A value zero however means that X_n is not affected (i.e., a real loss did not occur). We have,

$$\mathrm{P}\{U_n = 1\} = p_G, \qquad \mathrm{P}\{U_n = 0\} = 1 - p_G,$$
$$\mathrm{P}\{V_n = 1\} = p_B, \qquad \mathrm{P}\{V_n = 0\} = 1 - p_B.$$

We proceed for the analysis by looking at the evolution of the transmission rate between two potential loss moments,

$$\begin{aligned} X_{n+1} &= (1-U_n)X_n 1\{Y_n = G\} + U_n \nu X_n 1\{Y_n = G\} \\ &+ (1-V_n)X_n 1\{Y_n = B\} + V_n \nu X_n 1\{Y_n = B\} + \alpha D_n \\ &= (1 - \bar{\nu}U_n)\, X_n 1\{Y_n = G\} + (1 - \bar{\nu}V_n)\, X_n 1\{Y_n = B\} + \alpha D_n, \end{aligned} \tag{4.2}$$

where $\bar{\nu}$ is equal to $1 - \nu$. $1\{\mathrm{A}\}$ is the indicator function equal to 1 if expression A is true and to 0 otherwise. Using this recurrent equation, let us first focus on the convergence and stability of process X_n. Define the column vector,

$$\mathbf{X}_n = (X_n 1\{Y_n = G\}, X_n 1\{Y_n = B\})^T.$$

The convergence of $\mathbf{X}_n$ to a stationary regime implies that of X_n (and hence that of $X(t)$ given that the Markov chain is ergodic) since

$$X_n = X_n 1\{Y_n = G\} + X_n 1\{Y_n = B\}.$$

Define the matrix

$$Q_n = \begin{pmatrix} (1-\bar{\nu}U_n)1\{Y_{n+1} = G\} & (1-\bar{\nu}V_n)1\{Y_{n+1} = G\} \\ (1-\bar{\nu}U_n)1\{Y_{n+1} = B\} & (1-\bar{\nu}V_n)1\{Y_{n+1} = B\} \end{pmatrix}.$$

Finally, define the column vector

$$\mathbf{D}_n = (D_n 1\{Y_{n+1} = G\}, D_n 1\{Y_{n+1} = B\})^T.$$

Then it follows from (4.2)

$$\mathbf{X}_{n+1} = Q_n \mathbf{X}_n + \alpha \mathbf{D}_n. \tag{4.3}$$

This recurrent matrix equation is a particular case of the general stochastic linear difference equation $X_{n+1} = A_n X_n + B_n$, with A_n and B_n any two stationary ergodic processes [43, 67]. The next result follows from Theorem 2A in [67]. This theorem gives the required conditions for the process X_n to converge to a stationary regime independently of the initial state of the process. For this convergence to hold, we must have

(i) $-\infty \leq \mathrm{E}[\log |A_n|] < 0$

(ii) $\mathrm{E}[\log |B_n|^+] < \infty$

It is easy to prove that these conditions are satisfied by Equation (4.3). To this end, one can follow our approach in the Appendix in [11]. The following theorem follows.

Theorem 4.1 *Assume that the Markov chain* $\{Y_n\}$ *is initially in steady state. Consider an arbitrary initial state* $\mathbf{X}_0$*. Then,*

$$\mathbf{X}_n^* = \alpha \sum_{j=1}^{\infty} \left(\prod_{i=n-j}^{n-1} Q^i \right) \mathbf{D}_{n-j-1} \tag{4.4}$$

is the only solution of (4.3) and it converges absolutely almost surely. Furthermore, $|\mathbf{X}_n - \mathbf{X}_n^*| \to 0$ *a.s. for all* $\mathbf{X}_0$ *on the same probability space as* $\{(Q_n, \mathbf{D}_n)\}$*. In particular, the distribution of* $\mathbf{X}_n$ *converges to that of* $\mathbf{X}_n^*$ *as* $n \to +\infty$*.*

In summary, Theorem 4.1 says that the rate of the TCP connection X_n converges to a unique stationary regime defined by (4.4), and this is for any initial state X_0. This convergence is the result of two facts (which are summarized by the two conditions in Theorem 2A in [67]). First, the loss process by definition is stationary and of non-zero intensity. Second, due to the multiplicative decrease, the rate process gradually forgets its past (in fact the impact of past inter-loss times decreases geometrically fast). The stationarity of the process X_n and that of the loss process result directly in a stationarity of the process $X(t)$. In the following, we denote the stationary regime by *.

Next, we study the existence of moments of X_n in the stationary regime. We will use these moments when calculating the throughput. Note that the convergence of the process X_n in distribution does not imply that of its moments since the rate process $X(t)$ in our model is by definition not bounded. Later in this chapter, we add an approximation of the throughput in case of rate limitation (see Section 4.4). To prove the existence of moments of X_n, we define the following Laplace Stieltjis Transforms (LST) [80],

$$Z_n(s, G) = \mathrm{E}\left[e^{-sX_n} 1\{Y_n = G\}\right], \qquad Z_n(s, B) = \mathrm{E}\left[e^{-sX_n} 1\{Y_n = B\}\right],$$

with s a complex number having a positive real part. In addition, let us define

$$\mathbf{Z}_n(s) = \left[\; Z_n(s, G) \qquad Z_n(s, B) \; \right].$$

The k th moment of X_n $(k = 1, 2, \ldots)$ is related to $\mathbf{Z}_n(s)$ in the following way

$$\begin{aligned} \mathrm{E}\left[X_n^k\right] &= \mathrm{E}\left[X_n^k 1\{Y_n = G\}\right] + \mathrm{E}\left[X_n^k 1\{Y_n = B\}\right], \\ \mathbf{Z}_n^{(k)}(0) &= (-1)^k \left[\; \mathrm{E}\left[X_n^k 1\{Y_n = G\}\right] \qquad \mathrm{E}\left[X_n^k 1\{Y_n = B\}\right] \; \right]. \end{aligned} \tag{4.5}$$

The (k) exponent added to a function or a vector of functions denotes its kth derivative.

Theorem 4.2 *The Laplace Stieltjis Transform* $\mathbf{Z}_n(s)$ *satisfies the following recurrent equation,*

$$\mathbf{Z}_{n+1}(s) = D^*(\alpha s)\mathbf{Z}_n(s)P_1 + D^*(\alpha s)\mathbf{Z}_n(\nu s)P_2, \tag{4.6}$$

where

$$P_1 = \begin{bmatrix} \gamma(1-p_{_G}) & \bar{\gamma}(1-p_{_G}) \\ \bar{\beta}(1-p_{_B}) & \beta(1-p_{_B}) \end{bmatrix} \quad \textit{and} \quad P_2 = \begin{bmatrix} \gamma p_{_G} & \bar{\gamma} p_{_G} \\ \bar{\beta} p_{_B} & \beta p_{_B} \end{bmatrix}.$$

Proof: Using the recurrent Equation (4.2), we write

$$\begin{aligned} & E\left[e^{-sX_{n+1}}1\{Y_{n+1}=G\}\right] \\ &= \gamma E\left[e^{-s((1-\bar{\nu}U_n)X_n+\alpha D_n)}1\{Y_n=G\}\right] + \bar{\beta}E\left[e^{-s((1-\bar{\nu}V_n)X_n+\alpha D_n)}1\{Y_n=B\}\right] \\ & E\left[e^{-sX_{n+1}}1\{Y_{n+1}=B\}\right] \\ &= \bar{\gamma}E\left[e^{-s((1-\bar{\nu}U_n)X_n+\alpha D_n)}1\{Y_n=G\}\right] + \beta E\left[e^{-s((1-\bar{\nu}V_n)X_n+\alpha D_n)}1\{Y_n=B\}\right] \end{aligned}$$

Given that,

$$\begin{aligned} E\left[e^{-s(1-\bar{\nu}U_n)X_n}1\{Y_n=G\}\right] &= (1-p_{_G})Z_n(s,G) + p_{_G}Z_n(\nu s,G), \\ E\left[e^{-s(1-\bar{\nu}V_n)X_n}1\{Y_n=B\}\right] &= (1-p_{_B})Z_n(s,B) + p_{_B}Z_n(\nu s,B), \end{aligned}$$

we obtain the required relations. ■

Now with the help of recurrent Equation (4.6), we can investigate the convergence of moments $\mathrm{E}\left[X_n^k\right], k = 1, 2, \ldots$, for an arbitrary initial state X_0. First, we define the following limits

$$X^{(k)} = \lim_{n\to\infty} \mathrm{E}\left[X_n^k\right],$$

$$\mathbf{X}^{(k)} = \begin{bmatrix} X_{_G}^{(k)} & X_{_B}^{(k)} \end{bmatrix},$$

where

$$X_{_G}^{(k)} = \lim_{n\to\infty} \mathrm{E}\left[X_n^k 1\{Y_n=G\}\right], \qquad X_{_B}^{(k)} = \lim_{n\to\infty} \mathrm{E}\left[X_n^k 1\{Y_n=B\}\right]. \tag{4.7}$$

The existence of $X_{_G}^{(k)}$ and $X_{_B}^{(k)}$ implies directly the existence of $X^{(k)}$ (Equation 4.5). Furthermore,

$$X^{(k)} = X_{_G}^{(k)} + X_{_B}^{(k)}.$$

In the next theorem we formulate conditions for the existence of the limits in (4.7).

Theorem 4.3 *Let the first k moments of* D_n *exist. Then, the moments* $X_{_G}^{(k)}$ *and* $X_{_B}^{(k)}$ *exist and can be calculated from the following recurrent relation*

$$\mathbf{X}^{(k)} = \sum_{i=1}^{k} C_k^i \alpha^i d^{(i)} \mathbf{X}^{(k-i)}[P_1 + \nu^{k-i}P_2][I - P_1 - \nu^k P_2]^{-1}, \tag{4.8}$$

where $\mathbf{X}^{(0)} = [\pi_{_G}\ \pi_{_B}]$. *Finally,* $X^{(k)} = X_{_G}^{(k)} + X_{_B}^{(k)}$ *is the moment of the process* X_n *in the stationary regime.*

Proof: We first differentiate k times the recurrent relation (4.6) with respect to s,

$$\mathbf{Z}_{n+1}^{(k)}(s) = \sum_{i=0}^{k} C_k^i \alpha^i D^{*(i)}(\alpha s)\mathbf{Z}_n^{(k-i)}(s)P_1 + \sum_{i=0}^{k} C_k^i \alpha^i D^{*(i)}(\alpha s)\nu^{k-i}\mathbf{Z}_n^{(k-i)}(\nu s)P_2,$$

where C_k^i is the binomial coefficient equal to $k!/(i!(k-i)!)$. Then, we take $s = 0$ to get

$$\mathbf{Z}_{n+1}^{(k)}(0) = \sum_{i=0}^{k} C_k^i \alpha^i D^{*(i)}(0)\mathbf{Z}_n^{(k-i)}(0)[P_1 + \nu^{k-i}P_2]. \tag{4.9}$$

Let us now prove the convergence of $Z_n^{(k)}(0)$ when n increases. Once proven, we can let n go to infinity in (4.9) in order to get the expression of $\mathbf{X}^{(k)}$ given in (4.8). The last statement of the theorem follows immediately from (4.5). Recall that,

$$\mathbf{Z}_n^{(i)}(0) = (-1)^i \left[\begin{array}{cc} \mathrm{E}\left[X_n^i 1\{Y_n = G\}\right] & \mathrm{E}\left[X_n^i 1\{Y_n = B\}\right] \end{array} \right],$$

$$d^{(i)} = (-1)^i D^{*(i)}(0).$$

Let us introduce the following augmented vector

$$\Xi_n^{(k)} = \left[\begin{array}{cccc} \mathbf{Z}_n^{(0)}(0) & \mathbf{Z}_n^{(1)}(0) & \cdots & \mathbf{Z}_n^{(k)}(0) \end{array} \right].$$

Then, recursions (4.9) can be written in the following matrix form

$$\Xi_{n+1}^{(k)} = \Xi_n^{(k)}\Pi,$$

where

$$\Pi = \left[\begin{array}{ccccc} P & -\alpha d^{(1)}P & (-\alpha)^2 d^{(2)}P & \cdots & (-\alpha)^k d^{(k)}P \\ 0 & P_1 + \nu P_2 & C_2^1(-\alpha)d^{(1)}[P_1 + \nu P_2] & \cdots & C_k^{k-1}(-\alpha)^{k-1}d^{(k-1)}[P_1 + \nu P_2] \\ 0 & 0 & P_1 + \nu^2 P_2 & \cdots & C_k^{k-2}(-\alpha)^{k-2}d^{(k-2)}[P_1 + \nu^2 P_2] \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & & \cdots & P_1 + \nu^k P_2 \end{array} \right].$$

We used for the first line of matrix Π the fact that $P_1 + P_2 = P$. Given that by definition p_{B} is strictly positive and $\nu < 1$, all the matrices $[P_1 + \nu^k P_2], k \geq 1$, are sub-stochastic and hence they have eigenvalues with modulus less than one. Now, P is the transition matrix of an ergodic Markov chain. Therefore it only has one eigenvalue equal to one. Since the spectrum of Π is the union of spectrums of diagonal sub-matrices $[P_1 + \nu^k P_2]$ and P, we conclude that Π only has one eigenvalue equal to one and the other eigenvalues with modulus less than one. The latter implies that the powers of Π converge to the eigenprojection corresponding to the eigenvalue one. Consequently, $\mathbf{Z}_n^{(k)}(0)$ and the moments of X_n are also convergent. ■

Corollary 4.1 *Let $d = \mathrm{E}[D_n] < \infty$. Then,*

$$X_{G} = X_{G}^{(1)} = \alpha d \frac{\theta_{B}(\pi_{B} - \beta) + \pi_{G}}{1 - \theta_{B}\beta - \theta_{G}\gamma + \theta_{B}\theta_{G}(\gamma + \beta - 1)},$$

$$X_{_B} = X_{_B}^{(1)} = \alpha d \frac{\theta_{_G}(\pi_{_G} - \gamma) + \pi_{_B}}{1 - \theta_{_B}\beta - \theta_{_G}\gamma + \theta_{_B}\theta_{_G}(\gamma + \beta - 1)},$$

with $\theta_{_G} = 1 - \bar{\nu}p_{_G}$, $\theta_{_B} = 1 - \bar{\nu}p_{_B}$.

Corollary 4.2 *Let* $d = \mathrm{E}[D_n] < \infty$ *and* $d^{(2)} = \mathrm{E}\left[D_n^2\right] < \infty$. *Then*

$$X_{_G}^{(2)} = \frac{2\alpha d\theta_{_G}\theta_{_B}^{(2)}X_{_G}(1-\gamma-\beta) + 2\alpha d(\theta_{_B}X_{_B}\bar{\gamma} + \theta_{_G}X_{_G}\gamma) + \alpha^2 d^{(2)}(\theta_{_B}^{(2)}\pi_{_B} + \pi_{_G} - \beta\theta_{_B}^{(2)})}{\left(1 - \gamma\theta_{_G}^{(2)} - \beta\theta_{_B}^{(2)} - \theta_{_G}^{(2)}\theta_{_B}^{(2)}(1-\gamma-\beta)\right)},$$

$$X_{_B}^{(2)} = \frac{2\alpha d\theta_{_G}^{(2)}\theta_{_B}X_{_B}(1-\gamma-\beta) + 2\alpha d(\theta_{_G}X_{_G}\bar{\gamma} + \theta_{_B}X_{_B}\beta) + \alpha^2 d^{(2)}(\theta_{_G}^{(2)}\pi_{_G} + \pi_{_B} - \gamma\theta_{_G}^{(2)})}{\left(1 - \gamma\theta_{_G}^{(2)} - \beta\theta_{_B}^{(2)} - \theta_{_G}^{(2)}\theta_{_B}^{(2)}(1-\gamma-\beta)\right)},$$

with $\theta_{_G}^{(2)} = 1-(1-\nu^2)p_{_G}$ *and* $\theta_{_B}^{(2)} = 1-(1-\nu^2)p_{_B}$. $X_{_G}$, $X_{_B}$, $\theta_{_G}$ *and* $\theta_{_B}$ *are given in Corollary 4.1.*

4.2.1 Calculation of the throughput

Theorem 4.4 *Let* $d = \mathrm{E}[D_n] < \infty$ *and* $d^{(2)} = \mathrm{E}\left[D_n^2\right] < \infty$. *Then, the throughput of TCP on a two-state Markovian path is equal to*

$$\bar{X} = \lim_{t\to+\infty}\frac{1}{t}\int_0^t X(\tau)\mathrm{d}\tau = \theta_{_G}X_{_G} + \theta_{_B}X_{_B} + \frac{1}{2}\alpha\frac{d^{(2)}}{d},$$

where $X_{_G}$, $X_{_B}$, $\theta_{_G}$ *and* $\theta_{_B}$ *are given in Corollary 4.1 and where the second equality holds in the almost sure sense.*

Proof: $\{X_n\}$ forms an ergodic Markov chain since it converges in distribution and moments to a unique stationary regime. Hence, $\{t_n, X_n\}$ is an ergodic marked point process. From [44, Ch. 4] it follows that the associated continuous time process of the transmission rate evolution $X(t)$ is ergodic as well. The latter fact implies that the throughput, that is the time average transmission rate, is almost sure equal to $\mathrm{E}[X(t)]$, the expectation of the transmission rate at arbitrary time moment in the stationary regime. This expectation can be calculated by using the following inversion formula of the Palm theory (see e.g., [22, Ch.1, Sec.4])

$$\mathrm{E}[X(t)] = \frac{1}{d}\mathrm{E}^0\left[\int_0^{t_1} X(\tau)\mathrm{d}\tau\right]. \tag{4.10}$$

where $\mathrm{E}^0[.]$ is an expectation associated with Palm distribution. In particular, $P^0(t_0 = 0) = 1$. Palm distribution is no other than the distribution of the transmission rate upon potential loss moments in the stationary regime. The term between brackets on the right-hand side of (4.10) is no other than the area located between the process $X(t)$, the time axis, and the two instants t_0 and t_1, given that $t_0 = 0$ and that the process $X(t)$ is in its stationary regime at time 0.

Using (4.2) and (4.10), we write

$$\begin{aligned}
\bar{X} &= \frac{1}{d}\mathrm{E}^0\left[\int_0^{t_1}((1-\bar{\nu}U_0)X_0 1\{Y_0=G\}+(1-\bar{\nu}V_0)X_0 1\{Y_0=B\}+\alpha\tau)\mathrm{d}\tau\right] \\
&= \frac{1}{d}\mathrm{E}^0\left[(1-\bar{\nu}U_0)X_0 1\{Y_0=G\}D_0+(1-\bar{\nu}V_0)X_0 1\{Y_0=B\}D_0+\frac{1}{2}\alpha D_0^2\right] \\
&= \mathrm{E}^0\left[(1-\bar{\nu}U_0)X_0 1\{Y_0=G\}\right]+\mathrm{E}^0\left[(1-\bar{\nu}V_0)X_0 1\{Y_0=B\}\right]+\frac{1}{2}\alpha\frac{d^{(2)}}{d} \\
&= \theta_{_G}X_{_G}+\theta_{_B}X_{_B}+\frac{1}{2}\alpha\frac{d^{(2)}}{d}.
\end{aligned}$$

This concludes the proof of the theorem. ■

In a similar way to Theorem 4.4, we can prove the following result.

Theorem 4.5 *Let $d=\mathrm{E}[D_n]<\infty$, $d^{(2)}=\mathrm{E}\left[D_n^2\right]<\infty$ and $d^{(3)}=\mathrm{E}\left[D_n^3\right]<\infty$. Then, the second moment of the transmission rate can be expressed as*

$$\begin{aligned}
\bar{X}^{(2)} &= \lim_{t\to+\infty}\frac{1}{t}\int_0^t X^2(\tau)\mathrm{d}\tau \\
&= \mathrm{E}\left[X^2(t)\right]=\frac{1}{d}\mathrm{E}^0\left[\int_0^{t_1}X^2(\tau)\mathrm{d}\tau\right] \\
&= \theta_{_G}^{(2)}X_{_G}^{(2)}+\theta_{_B}^{(2)}X_{_B}^{(2)}+\theta_{_G}X_{_G}\alpha\frac{d^{(2)}}{d}+\theta_{_B}X_{_B}\alpha\frac{d^{(2)}}{d}+\frac{1}{3}\alpha^2\frac{d^{(3)}}{d},
\end{aligned}$$

where $X_{_G}^{(2)}$, $X_{_B}^{(2)}$, $\theta_{_G}^{(2)}$ and $\theta_{_B}^{(2)}$ are given in Corollary 4.2 and where the second equality holds in the almost sure sense.

The second moment is also of particular interest since it tells us how much the transmission rate varies. A small variation is necessary for a good quality in real time applications.

4.3 Impact of burstiness of losses

We study in this section, theoretically and with simulations, the impact of a fluctuation of the loss rate on the throughput of a TCP connection. In particular, we show how much the throughput of TCP changes when loss events tend to appear in bursts. This is a first try to understand the impact of a change in the distribution of losses on the performance of TCP congestion control. The understanding will be completed in the next chapter when we present our general model for losses. Our study will also tell us how well a model only considering the average loss rate predicts the throughput of TCP on paths where loss events are clustered. See Figure 3.4 for an example of window evolution in presence of clustered loss events. To accomplish such a study, we consider the following particular case of our previous model,

$$p_{_G}=0, \qquad p_{_B}=1.$$

In other words we suppose that if the path is in the Bad state, all potential losses are transformed into real losses, and if it is in the Good state no real losses occur. This model is sufficiently general to allow to vary both the average loss rate as well as the burstiness of losses. We set ν in the analysis to 0.5. Substituting in the expressions of X_G and X_B (Corollary 4.1), we get

$$X_B = 2\alpha d, \qquad X_G = \alpha d \frac{\bar{\beta} + \pi_G}{\bar{\gamma}}. \tag{4.11}$$

The throughput in this particular case is given by,

$$\bar{X} = X_G + \frac{1}{2} X_B + \frac{1}{2} \alpha \frac{d^{(2)}}{d}. \tag{4.12}$$

Remark 4.1 *It may seem remarkable that X_B given by Equation (4.11) does not depend on the transition probabilities of the Markov chain. This can be easily explained using the following argument. The mean time between losses is clearly $1/\lambda = d/\pi_B$, so the mean increase in TCP rate X between two consecutive losses is $\alpha d/\pi_B$. Since we assume that we are in the stationary regime, the mean decrease in X between losses should be equal to the mean increase. But, the mean decrease in X is half its mean value at loss. Thus,*

$$\mathrm{E}\left[X_n^* | Y_n = B\right] = 2\alpha d/\pi_B.$$

We conclude that indeed,

$$X_B = \mathrm{E}\left[X_n^* 1\{Y_n^* = B\}\right] = \mathrm{E}\left[X_n^* | Y_n^* = B\right] \mathrm{P}\left\{Y_n^* = B\right\} = 2\alpha d.$$

4.3.1 The reference throughput

To study the impact of burstiness of losses, we change the parameters of the Markov chain (β and γ) while keeping the average loss rate λ unchanged. We then compare the throughput in the bursty case to that achieved when the connection sees a non-bursty loss process with the same rate. We denote the latter throughput by $\bar{X}_r$ and we use it as a reference to evaluate the impact of burstiness of losses.

A non-bursty loss process is obtained when the loss probability upon potential loss moments is the same in both states. We call this probability p. To get the same average loss rate as in the bursty case, p must be equal to

$$p = d\lambda = p_G \pi_G + p_B \pi_B = \pi_B.$$

Lemma 4.1 *On a non-bursty path, the source achieves a throughput equal to*

$$\bar{X}_r = \frac{2-p}{p} \alpha d + \frac{1}{2} \alpha \frac{d^{(2)}}{d}. \tag{4.13}$$

Proof: This expression of $\bar{X}_r$ can be easily obtained by substituting θ_G and θ_B in the general expression of $\bar{X}$ (Theorem 4.4) by their values as a function of p, the drop probability in the two states. We have in the non-bursty case

$$\theta_G = \theta_B = 1 - \frac{p}{2}.$$

The parameters of the Markov chain disappear and we get the above nice expression of the reference throughput as a function of p and the distribution of potential losses. ■

4.3.2 Variation of the throughput with burstiness

The non-bursty path that has the same average loss rate is obtained when taking a loss probability p equal to π_B in both states. Using (4.13), the reference throughput in the non-bursty case is then equal to

$$\bar{X}_r = \frac{2\alpha}{\pi_B} - \alpha d + \frac{1}{2}\alpha\frac{d^{(2)}}{d}.$$

We express $\bar{X}$ as a function of $\bar{X}_r$ and the parameters of the Markov chain. The expression of X in the bursty case is given by (4.11) and (4.12). We get

$$\bar{X} = \bar{X}_r + \alpha d \pi_G \left[\frac{1}{\bar{\gamma}} - \frac{1}{\pi_B}\right]. \tag{4.14}$$

It is clear from this expression of $\bar{X}$ that the non-bursty case is obtained when $\bar{\gamma} = \beta = \pi_B$. In our particular case ($p_G = 0, p_B = 1$), $\bar{\gamma}$ is the probability that the next potential loss causes a real loss given that we are in the Good state. β is the probability that it causes a loss given that we are in the Bad state. In the non-bursty case, these two probabilities must be equal. At the same time, they must be equal to π_B, the probability that the next potential loss causes a real loss independently of the current state.

We fix now d and we increase β and γ in such a way that their ratio remains constant. This guarantees that π_B and π_G, and hence the average loss rate λ, remain the same. in fact, the increase in β and γ stretches the duration of the Good and Bad states which makes the path of the connection more bursty. The reference throughput remains unchanged since it is only a function of the average loss rate and the process of potential losses. Hence, Equation (4.14) shows that the throughput $\bar{X}$ improves with the increase in burstiness we introduced. Theoretically, the throughput of TCP goes to infinity when γ approaches one. This is because the Good state becomes of infinite duration. Practically, this is not possible since the rate cannot keep increasing for a very long time without the occurrence of any congestion event. There is always an intrinsic upper limit on the burstiness of a loss process.

4.3.3 Simulation-based validation of the model

We validate our model using the TCP implementation in the `ns` simulator [102]. We consider long TCP transfers and we use the SACK version of TCP [56] since it is able to recover quickly

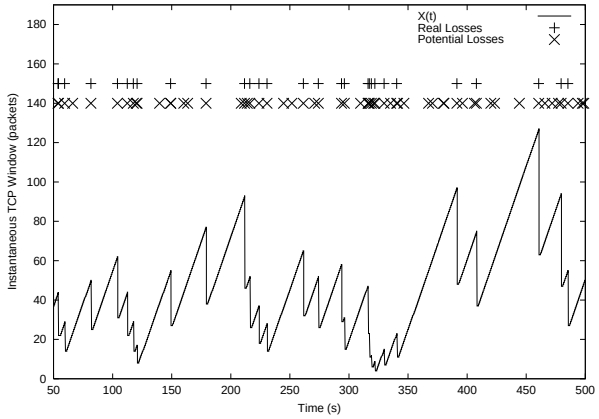


Figure 4.3: Simulation: Variation of $X(t)$ vs. time on a Markovian path

from losses and with a low probability of timeout and slow start. We suppose that the receiver acknowledges every data packet ($b = 1$). For the moment, we assume that the receiver window is very large so that it does not affect the growth of TCP transmission rate.

The simulation scenario consists of a TCP connection crossing a 2 Mbps link. The round-trip time of the connection is taken equal to 560 ms. TCP packets are of total size 1000 bytes. We add our Markovian loss model to the simulator and we associate it to the 2 Mbps link. We only account for losses on the 2 Mbps link and we evaluate their impact on the throughput. We choose the parameters of the simulation in a way not to get packet losses in the other parts of the network. This clearly requires that losses on the 2 Mbps link are frequent so that the rate of TCP remains low and the buffers in network routers do not overflow. The purpose of the present simulation is just to validate our result on the increase of the throughput with burstiness. Later in this chapter, we will show how our model can be used to predict the throughput of real TCP connections and we will validate our results in real scenarios.

The time between potential losses is taken to be exponentially distributed. Figure 4.3 shows a typical example of the variation of TCP window in the presence of our Markovian model for losses. We see well in the figure how potential losses are transformed into real losses and how real losses cause a reduction of the window by a factor 0.5. We then run the connection for one hour and we calculate the values of X_G, X_B and $\bar{X}$. The simulation results are compared to those given by our analysis. We transform all the results of our analysis from rate to window since the calculation of moments of window in practice is easier than the calculation of moments of rate (except for the throughput). When simulating, X_G (resp. X_B) is calculated by summing the window sizes when potential losses occur and the link in the Good state (resp. in the Bad state), then by dividing this sum by the total number of potential losses. $\bar{X}$ is calculated as the throughput of the connection over one hour expressed in terms of packets/s times RTT. This gives the time average of the congestion window.

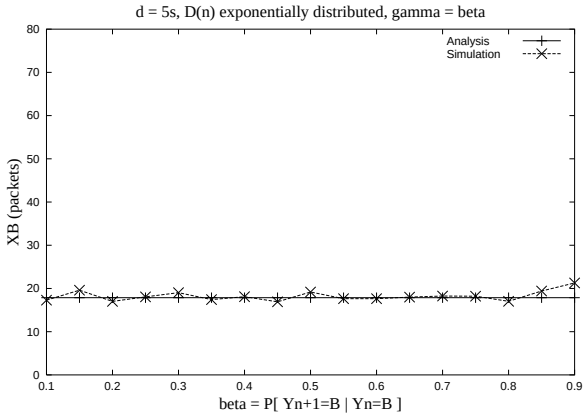


Figure 4.4: X_B vs. burstiness

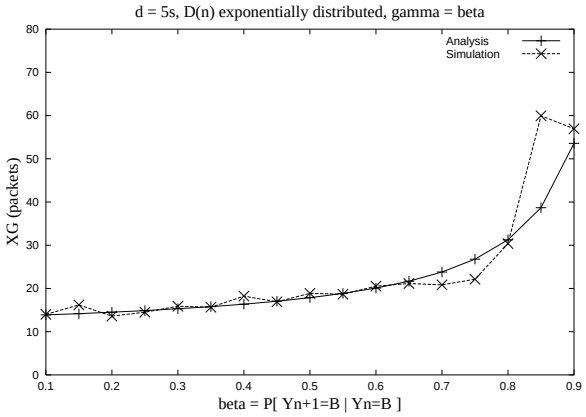


Figure 4.5: X_G vs. burstiness

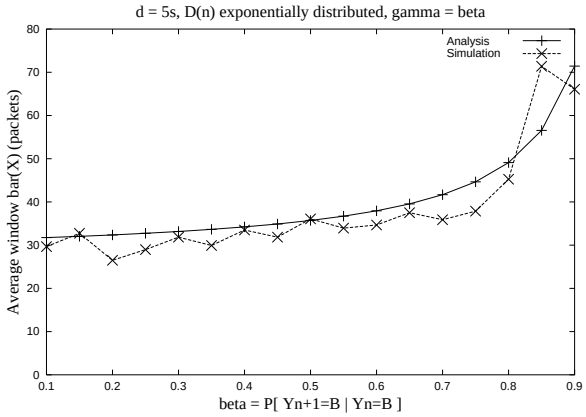


Figure 4.6: $\bar{X}$ vs. burstiness

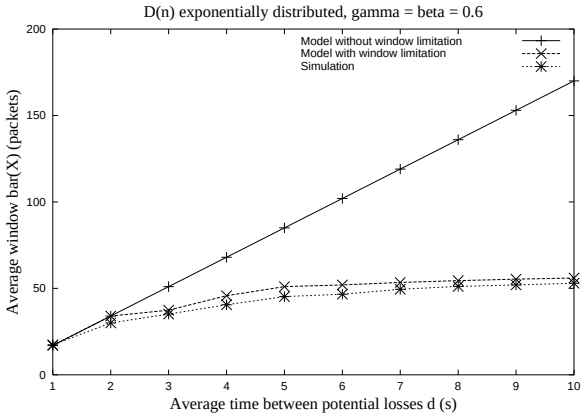


Figure 4.7: $\bar{X}$ vs. d with rate limitation

Now, we fix the average time between potential losses (d) to 5 seconds and we increase the transition probabilities β and γ while keeping $\beta = \gamma$. This results in $\pi_G = \pi_B = 0.5$ and a constant average loss rate $\lambda = 0.1$s. Our analysis tells us that X_B must not change (Equation (4.11)). X_G and $\bar{X}$ must however increase as a result of the increase in burstiness (Equations (4.11) and (4.14)). Figures 4.4, 4.5 and 4.6 validate our results. In particular, it is clear from Figure 4.6 that by increasing β from 0.1 to 0.9, the throughput of TCP increases by about 100% even though the average loss rate is the same. This confirms our main finding concerning the improvement of the performance of TCP when losses become clustered.

4.4 Case of window limitation

Our previous model does not account for the case when the receiver window limits the growth of the congestion window. As we explained in Section 3.2.3, it is difficult to calculate exactly the throughput in presence of such a limitation given that the model becomes sub-linear. Even

for a simple Poisson process the solution is not so evident as we will see in Chapter 6. Now, for a complicated loss process as the one in this chapter, some approximation of the throughput can always be found. A typical approximation is the fixed-point one [105] where the congestion window is assumed to always reach the maximum window between two real loss events. We show in this section how such an approximation can be applied to our model. For simplicity of the presentation, we consider the particular case of the previous section, that is $\nu = 0.5$, $p_G = 0$ and $p_B = 1$. We will profit from the fact that the path presents two states to get a better approximation of the throughput than when the path is assumed to only have one state [105].

The first thing to do is to write conditions on the loss process that define the region where our previous model without window limitation can be used. Once this region is defined, all what we still need to do is to find approximations of the throughput in the other regions. Denote the upper bound for the transmission rate by M. It is equal to the receiver window divided by RTT. Consider our previous model where the rate varies without any limitation. The point where the transmission rate is most likely to reach the maximal value M is just before the first potential loss in the Bad state. This corresponds to the first reduction of the transmission rate after getting out of the Good state. For our previous model to work, the expectation of the transmission rate at this point must be much smaller than the upper bound M. This condition can be written as,

$$\mathrm{E}\left[X_n^* | Y_n^* = B, Y_{n-1}^* = G\right] << M.$$

Taking into account that

$$\begin{aligned} \mathrm{E}\left[X_n^* | Y_n^* = B, Y_{n-1}^* = G\right] &= \mathrm{E}\left[X_n^* | Y_{n-1}^* = G\right] \\ &= \mathrm{E}\left[X_{n-1}^* | Y_{n-1}^* = G\right] + \alpha d = X_G/\pi_G + \alpha d, \end{aligned}$$

we get the following condition

$$X_G/\pi_G + \alpha d << M,$$

where X_G is given by (4.11). Intuitively, the larger the average loss rate and the lower the burstiness, the more likely that this condition holds. For a given loss rate, the increase in burstiness stretches the duration of the Good state and makes more likely that the transmission rate reaches the upper bound at its end. Now, the closer $\mathrm{E}\left[X_n^* | Y_n^* = B, Y_{n-1}^* = G\right]$ to M, the more important the impact of the receiver window. First, the receiver window starts to impact the rate evolution during the Good state. The rate evolution during the Bad state can be assumed to be close to that predicted by our model without window limitation. The receiver window starts to impact the two states of the path once the expectation of the transmission rate just prior to losses in the Bad state becomes larger than M. This latter condition can be written as

$$\mathrm{E}\left[X_n^* | Y_n^* = B, Y_{n-1}^* = B\right] >> M, \qquad \text{i.e.,} \qquad \frac{1}{2}\frac{X_B}{\pi_B} + \alpha d >> M.$$

Now we pass to the approximation of the throughput when the receiver window impacts one of the two states. We use for this purpose a fixed-point approach similar to that in [105]. We assume that once the upper bound starts to impact a state, the rate always reaches its maximal value. Using the above two conditions, we separate the space into three regions. In the first region, the transmission rate is not affected by M. In the second region, the Good state is affected. In the third region, both states are affected. We then use the above assumption to approximate the throughput of the transfer during each state of the path. Let

$$\bar{X}_{G} = \mathrm{E}\left[X(t)|Y(t) = G\right], \qquad \bar{X}_{B} = \mathrm{E}\left[X(t)|Y(t) = B\right],$$

where the expectation is with respect to the stationary probability. Thus, the overall throughput is simply equal to

$$\bar{X} = \mathrm{E}\left[X(t)\right] = \pi_{G}\bar{X}_{G} + \pi_{B}\bar{X}_{B}. \tag{4.15}$$

Let us now approximate the throughput in each of the three regions we defined:

$\mathrm{E}\left[\mathbf{X_n^*}|\mathbf{Y_n^*} = \mathbf{B}, \mathbf{Y_{n-1}^*} = \mathbf{G}\right] < \mathbf{M}$**:** The receiver window in this case has no influence on the rate evolution and the throughput given by Equation (4.12) can be considered.

$\mathrm{E}\left[\mathbf{X_n^*}|\mathbf{Y_n^*} = \mathbf{B}, \mathbf{Y_{n-1}^*} = \mathbf{G}\right] > \mathbf{M}$ **but** $\mathbf{E}[\mathbf{X_n^*}|\mathbf{Y_n^*} = \mathbf{B}, \mathbf{Y_{n-1}^*} = \mathbf{B}] < \mathbf{M}$**:** During the Bad state the receiver window has no impact and $\bar{X}_{B}$ can be simply approximated by taking $p = 1$ in Equation (4.13). This is the throughput obtained when the transmission rate is reduced at every potential loss and when the window is not limited, which is the case for the Bad state in this region. Thus,

$$\bar{X}_{B} = \alpha d + \frac{1}{2}\alpha\frac{d^{(2)}}{d}.$$

During the Good state, the throughput can be approximated using the fixed-point approach. On average, the rate at the beginning of the Good state is equal to

$$X_0 = \mathrm{E}\left[X_n^*|Y_n^* = G, Y_{n-1}^* = B\right] = \frac{1}{2}\frac{X_{B}}{\pi_{B}} + \alpha d.$$

The average duration of the Good state equals $d/\bar{\gamma}$. Using our assumption that M is always reached during the Good state in this region, we consider that the transmission rate grows first from X_0 to M, then stays at M until the beginning of the Bad state. This gives us the following expression for $\bar{X}_{G}$,

$$\begin{aligned}\bar{X}_{G} &= \frac{\bar{\gamma}}{d}\left(\int_{0}^{(M-X_0)/\alpha}(X_0 + \alpha t)\mathrm{d}t + \int_{(M-X_0)/\alpha}^{d/\bar{\gamma}} M\mathrm{d}t\right)\\ &= \frac{\bar{\gamma}}{d}\left(\frac{M^2 - X_0^2}{2\alpha} + M\left(\frac{d}{\bar{\gamma}} - \frac{M - X_0}{\alpha}\right)\right).\end{aligned}$$

Given $\bar{X}_{G}$ and $\bar{X}_{B}$, the throughput in this region can be approximated using Equation (4.15).

$\mathbf{E[X^*_n|Y^*_n = B, Y^*_{n-1} = B] > M}$: The fixed-point approach tells us that in this region, the transmission rate always reaches M between potential losses. The rate of TCP just before the occurrence of a real loss can be taken equal to M. Thus, $X_0 = M$ and we can write

$$\begin{aligned} \bar{X}_G &= M \\ \bar{X}_B &= \frac{1}{d}\left(\int_0^{M/2\alpha} (M/2 + \alpha t)\,\mathrm{d}t + \int_{M/2\alpha}^{d} M\mathrm{d}t\right) = M - \frac{M^2}{8\alpha d}. \end{aligned}$$

The total throughput can be approximated by

$$\bar{X} = \pi_G \bar{X}_G + \pi_B \bar{X}_B = M - \frac{M^2 \pi_B}{8\alpha d}.$$

One of the advantages of our approximation compared to that in [105] is that we used the Markov chain of the path to define three regions instead of two. Indeed, if the path is assumed to only have one state, all that we can do is to condition on $\mathrm{E}[X^*_n]$. This is the kind of approximation used in [105]; the receiver window impacts the rate evolution if $\mathrm{E}[X^*_n]$ is greater than M and does not impact it if not. Increasing the number of states increases the number of regions (or conditions) which should result in a better approximation of the throughput.

We validate now our approximation of the throughput using our previous simulation scenario. We set β and γ to 0.6 (this gives $\pi_B = \pi_G = 0.5$) and we consider that the times between potential losses are exponentially distributed. We set the round-trip time of the connection to 250 ms and we take the receiver window equal to the bandwidth-delay product. We increase d from 1s to 10s and we plot in the same figure (Figure 4.7) the throughputs from simulation, from our model without window limitation and from our approximation. By a simple calculation, we can see that with this values we chose for the simulation, we cross the three regions we defined at the beginning of this section. The figure shows how our approximation gives close results to the simulated throughput. The model without window limitation leads to a clear overestimation.

4.5 Application of the model to real connections

We explain in this section how our Markovian model can be used to predict the throughput of a real TCP connection. Of course, one should expect that our model is more useful on paths where the loss rate fluctuates. The calculation of the throughput (Theorem 4.4) requires the knowledge of the parameters of the Markov chain associated to the path, the process of potential losses and the drop probabilities in the different states. We present a technique to infer these parameters from the trace of the connection [14]. We assume that we only have the moments at which the connection divides its window. This means that we have a realization of the process of times between congestion events $\{S_n\}$. From this realization we infer how the loss rate fluctuates. We assume that the path of the connection presents some kind of burstiness that justifies the use of

a Markov chain. We consider our LAN connection for this purpose. Recall that the loss process on the LAN connection is highly bursty (Section 3.3.1). We present our technique for the case of a two-state Markov chain. In the same way, the parameters of a multi-state Markovian model can be inferred from the trace of the connection.

Let us first estimate P, the transition matrix for the Markov chain $\{Y_n\}$. Recall that this is the Markov chain obtained when looking at the state of the path at potential loss moments. We need to determine when the path is in the "Good" state and when it is in the "Bad" state. We use the following simple method. Choose some time interval τ. We explain later how to make this choice. If the inter-loss time is less than τ, the path is assumed to be in the Bad state, otherwise it is assumed to be in the Good state. If two or more consecutive inter-loss times correspond to the same state, we merge them together and we call the new interval S_k^G or S_k^B depending on the state. Note that these new intervals represent the time during which the path of the connection is either in the Good or in the Bad state. Denote by N_G (resp. N_B) the number of intervals S_k^G (resp. S_k^B) during the lifetime of the connection. Then, the evolution of the path can be described by a two-state continuous-time Markov process with the following infinitesimal generator matrix [80]

$$Q = \begin{bmatrix} -\sigma_G & \sigma_G \\ \sigma_B & -\sigma_B \end{bmatrix}, \tag{4.16}$$

where the rates σ_G and σ_B are calculated as follows

$$\sigma_G = \frac{1}{\mathrm{E}\left[S_k^G\right]} \simeq \frac{N_G}{\sum_{k=1}^{N_G} S_k^G}, \quad \sigma_B = \frac{1}{\mathrm{E}\left[S_k^B\right]} \simeq \frac{N_B}{\sum_{k=1}^{N_B} S_k^B}.$$

Note that on some paths, say a wireless link, this continuous time Markov chain is *a priori* known (using the Rayleigh model for example [113]) and can be directly used without the need to look at the trace of the TCP connection. In case it is not known, we have to define it by separating the inter-loss times into two groups using the parameter τ as described above. We present now two approaches for the determination of τ. The first one is more empirical. It consists in looking at the distribution of inter-loss times (e.g., Figure 3.8) and choosing τ in order to separate the two distributions it encloses: the distribution of inter-loss times in the Good state and that of inter-loss times in the Bad state. For example, the distribution of times between losses on the LAN connection shows clearly a pulse around the origin (Figure 3.8). This pulse corresponds to losses that appear close to each other or in other words in bursts. The rest of the distribution corresponds to times between bursts. Bursts of losses can be better seen in Figure 3.4. We can assume that bursts of losses appear when the path is in the Bad state and choose τ in a way to isolate the pulse ($\tau \simeq 0.4$s).

The second method for the choice of τ is less empirical and was used in the context of Markov-Modulated Poisson Processes [93]. The parameter τ is taken equal to the expectation of inter-loss times, that is

$$\tau = \mathrm{E}\left[S_n\right] \simeq \frac{1}{N} \sum_{k=1}^{N} S_k,$$

where N is the total number of inter-loss intervals we get from the measurement.

Given the infinitesimal generator matrix of the continuous time Markov chain associated to the path (Q), we calculate the transition matrix P of the discrete time Markov chain embedded at the moments of potential losses. To this end, we use the uniformization technique [126]. Let us choose the potential loss process $\{D_n\}$ as a Poisson process of intensity $1/d$ higher than both $\sigma_{_G}$ and $\sigma_{_B}$. A reasonable choice of d is the average round-trip time. According to the uniformization technique [126], the state of a path described by the continuous time Markov process (4.16) and sampled at the moments of potential losses can be equivalently given by a discrete time Markov chain with the following transition matrix,

$$P = \begin{bmatrix} 1 - d\sigma_{_G} & d\sigma_{_G} \\ d\sigma_{_B} & 1 - d\sigma_{_B} \end{bmatrix}.$$

Having chosen d and calculated $\sigma_{_G}$ and $\sigma_{_B}$ from the trace, we can easily deduce the parameters β and γ of the loss model (Figure 4.2). Namely, $\bar{\gamma} = d\sigma_{_G}$ and $\bar{\beta} = d\sigma_{_B}$. We still have to determine $p_{_G}$ and $p_{_B}$. Let ω_k^G (resp. ω_k^B) be the number of real losses in the time interval S_k^G (resp. S_k^B). Then, the probabilities $p_{_G}$ and $p_{_B}$ can be approximated by

$$p_{_G} = \frac{\sum_{k=1}^{N_G} \omega_k^G}{\sum_{k=1}^{N_G} S_k^G/d} = \frac{d\sum_{k=1}^{N_G} \omega_k^G}{\sum_{k=1}^{N_G} S_k^G} = d\lambda_{_G}, \quad p_{_B} = \frac{\sum_{k=1}^{N_B} \omega_k^B}{\sum_{k=1}^{N_B} S_k^B/d} = \frac{d\sum_{k=1}^{N_B} \omega_k^B}{\sum_{k=1}^{N_B} S_k^B} = d\lambda_{_B}.$$

Note here that the choice of the intensity of potential (d) losses impacts the values of the drop probabilities. The more the potential loss moments, the smaller the drop probabilities. Different values can be given to d. However, we must make sure that intensity of potential losses is much higher than the intensity of real losses in both the Bad and the Good states. Decreasing the value of d improves the precision of the uniformization technique.

We consider the LAN connection described in Section 3.1 for the validation of our Markovian model. We take $\tau = 0.4$s for the separation of the Bad state from the Good state. For each trace file, we infer the different parameters of the model. In Figure 4.8, we compare the throughput given by our Markovian model to that of ideal TCP or the so-called exact fluid model. Recall that the ideal throughput is numerically calculated for a TCP version that increases its window linearly as a function of time (see Section 3.4). We plot in the same figure the throughputs obtained when the loss process is assumed to be Poisson and when it is assumed to be deterministic. For these two latter throughputs we use the formulas we find in the next chapter. It is clear that the Markovian model gives the best result when deterministic and Poisson losses don't work. However, it overestimates the ideal throughput when the Poisson process works. This is due to our choice of τ. Indeed, when the Poisson process does not work, the distribution of inter-loss times presents an important pulse close to the origin and this pulse corresponds to a second distribution that needs to be separated. Separating this distribution improves the performance with respect to a model that assumes that this pulse does not exist and that losses follow a Poisson process. Now, when the Poisson process works, the distribution of inter-loss times is

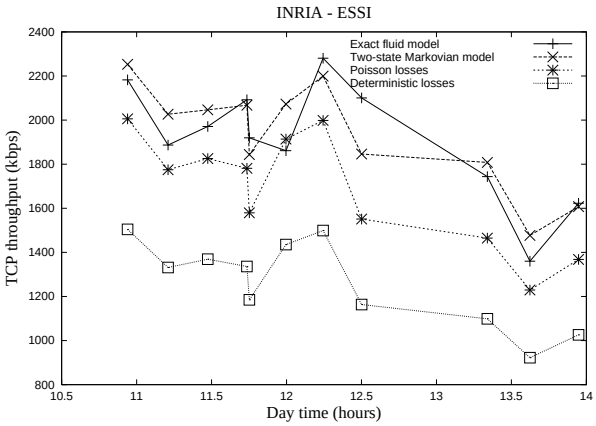


Figure 4.8: LAN: Validation of the two-state Markovian model

close to that of an exponential distribution and there is not another distribution that needs to be separated. However, our choice of τ results in a splitting of the exponential distribution into two non-exponential ones which results in this deterioration of performance. Note here that when taking the process of potential losses as Poisson and the drop probabilities as constant, we are making indirectly the assumption that the process of real losses is Poisson in both states, of course with different rates. This assumption holds when the loss process is highly bursty and does not hold when it is close to Poisson.

4.6 Conclusions

In this chapter we evaluated the impact of a fluctuation in the loss rate on the performance of TCP congestion control. In particular, we studied how TCP behaves on paths presenting some Good and Bad states. We defined a model for the network using potential losses and a two-state Markov chain. We then calculated the throughput and the moments of the transmission rate at potential loss moments. The throughput is compared to the one achieved when operating over a non-bursty path having the same average loss rate. Our main result is that for a given loss rate, the performance of TCP improves when losses tend to appear in bursts. We run a set of simulations with `ns` to validate the analytical results. We also presented a technique for the calculation of the parameters of the model (transition matrix of the Markov chain, drop probabilities) and hence the throughput of TCP.

We still have to refine our method for the separation of the different states of the path. Also, we still have to validate our model on paths oscillating according to an intrinsic Markov chain (e.g., wireless links). Another important issue to explore is the extension of the model to the case when the distributions of inter-loss times in the different states are not of the same type. In our previous model, these distributions are assumed to be of the same type since the process of potential losses is the same in the all states. A possible solution could be the use of Markov

Renewal Processes [59, 73] where the distributions of times between losses (arrivals in queueing terminology) change with the state of the path and where the state of the path changes according to a Markov chain.

Chapter 5

Modeling TCP congestion control: A general approach

Our previous Markovian model covers some but not all the loss processes that a TCP connection may see in the Internet. For example, it does not cover the case when the distribution of inter-loss times does not follow the same law in different states of the path. Moreover, the previous model does not show the impact of some parameters of the loss process, as the variance and the correlation, on the performance of TCP congestion control (yet it shows the impact of other parameters and allows easy derivation of all moments of the transmission rate). All these questions are solved with the general model we present in this chapter. The present model is sufficiently general that it can cover many of existing models of TCP. It can be considered as a general framework for TCP modeling. One can imagine any loss process, calculate some functions of this process, plug these functions in the expression of the throughput we will provide, and get the throughput of TCP. Our explicit expression of the throughput tells us clearly how the throughput varies with the distribution of losses and how well the throughput is estimated when we make a certain assumption on the loss process. We write the throughput as a function of the packet drop probability which yields a generalization of the square root formula [92, 105] for TCP throughput. Recall that the expression of the throughput is found explicitly for the case of no limitation on the window. Bounds for the throughput are found in the case of window limitation. Finally, our general expression of the throughput is specified to some particular loss processes, mainly to Markovian Arrival Processes (MAP) [20, 100]. The analysis in this chapter as well as the validation of the results can be found in [12].

5.1 The model

As in the previous chapter, we assume that the rate of the TCP connection increases continuously and linearly with time at a constant rate α between congestion events, and decreases multiplicatively with a factor $\nu \in (0, 1)$ upon congestion. Recall that α is approximately equal to $1/(bRTT^2)$, where RTT is the average round-trip time and b the number of data packets cov-

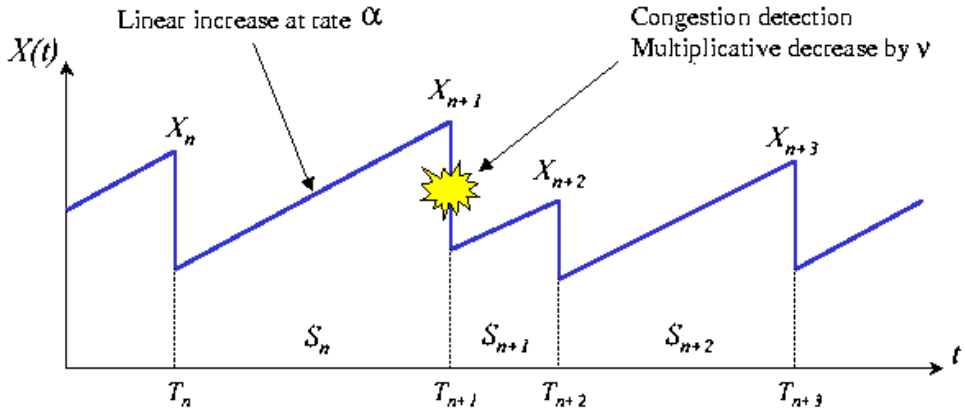


Figure 5.1: Variation of TCP rate on a general path

ered by an ACK. We work directly with the process of inter-loss times $\{S_n\}$ and we eliminate the intermediate potential loss process of our Markovian model. Assume for the moment that there is no limitation on the congestion window. Bounds for the throughput will be later introduced for the case of window limitation. Assume also that there is no timeout intervals. We substitute the throughput $\bar{X}$ obtained with this model in Equations (3.1) and (3.2) in order to account for timeouts and the packet nature of TCP.

Let T_n denote the moment at which the nth congestion event occurs, so that $S_n = T_{n+1} - T_n$. We look at the moments of congestion and we denote by X_n the transmission rate just prior to the nth congestion event. Thus, the transmission rate $X(t)$ can be described by the following recurrent equation,

$$X_{n+1} = \nu X_n + \alpha S_n. \tag{5.1}$$

Figure 5.1 shows the variation of the rate of the fluid model for a realization of the loss process. The pair $\{T_n, X_n\}$ can be considered as a marked point process [22]. Each point of the process, which corresponds to a congestion event or a loss event, is marked by the rate of the TCP connection upon its occurrence.

Concerning the loss process, we only make the assumption that it is stationary, ergodic and of non-null finite intensity. With this minor assumption, our model is able to capture any correlation and any distribution of inter-loss times. The loss process $\{S_n\}$ is characterized by the following functions:

- $s = 1/\lambda = \mathrm{E}[S_n]$: Average inter-loss time.
- $s^{(2)} = \mathrm{E}\left[S_n^2\right]$: Second moment of inter-loss times.
- $R(k) = \mathrm{E}[S_n S_{n-k}], k = 0, 1, \ldots$: Correlation function of order k. Particularly, $R(0) = s^{(2)}$.
- $C(k) = R(k) - s^2, k = 0, 1, \ldots$: Covariance function of order k. Particularly, $C(0)$ is equal to V, the variance of inter-loss times.

We also use the " ˆ " exponent to denote the result of the normalization of a function to some appropriate power of the average inter-loss time s. In particular, for $k = 0, 1, \ldots$, $\hat{C}(k) = C(k)/s^2$, $\hat{R}(k) = R(k)/s^2$ and $\hat{V} = V/s^2$. Using these functions, we find in the following the explicit expression of TCP throughput as well as the first two moments of the transmission rate (or the window).

5.2 Performance analysis

Equation (5.1) is again a particular case of the general stochastic linear difference equation $X_{n+1} = A_n X_n + B_n$ [43]. Since the inter-loss time process is stationary ergodic, and since $\nu < 1$ and $\mathrm{E}[S_n] < \infty$, it follows from Theorem 2A in [67] (see the Appendix in [11] for more details) that Equation (5.1) has a unique stationary solution given by,

$$X_n^* = \alpha \sum_{k=0}^{\infty} \nu^k S_{n-1-k}. \tag{5.2}$$

Whatever is its initial state, the rate (or the window) process converges almost sure to the above stationary regime [43, 67],

$$\lim_{n\to\infty} |X_n - X_n^*| = 0, \quad P - \text{a.s.}$$

We denote the unique stationary regime of the rate of the TCP connection by " * ". Again, this convergence is the result of the linear-increase and the multiplicative-decrease which make the rate forget asymptotically its past. In the stationary regime, the transmission rate of the TCP connection at any moment can be expressed as only a function of the reduction factor ν, the slope α and the sum of inter-loss times in the past. This is what Equation (5.2) expresses. We also see in this equation how the impact of inter-loss times in the past disappears quickly given the geometrical decrease in the coefficient ν^k. The result will be a limited impact of the correlation of losses on the performance of TCP.

The fact that the stationary regime is unique and independent of the initial state can be considered as a proof of the conditional fairness of linear-increase multiplicative-decrease congestion control policies. If all the flows use the same parameters ν and α, and if they are subject to the same loss process (i.e., if they reduce their rates at the same instants), the network will converge to a stationary regime where the flows get the same throughput according to Equation (5.2) whatever are their initial rates. This conditional fairness has been proved in a similar way in [50].

5.2.1 Calculation of the first two moments of X_n

Proposition 5.1 *The first two moments of the rate process in the stationary regime are given by*

$$\mathrm{E}[X_n^*] = \frac{\alpha}{\lambda(1-\nu)}, \tag{5.3}$$

$$\mathrm{E}\left[(X_n^*)^2\right] = \frac{\alpha^2}{1-\nu^2}\left[R(0) + 2\sum_{k=1}^{\infty}\nu^k R(k)\right]. \tag{5.4}$$

Proof: To calculate (5.3) and (5.4), we use the expression (5.2) for the stationary regime.

$$\mathrm{E}\left[X_n^*\right] = \alpha\sum_{k=0}^{\infty}\nu^k \mathrm{E}\left[S_{n-1-k}\right] = \frac{\alpha}{\lambda}\sum_{k=0}^{\infty}\nu^k = \frac{\alpha}{\lambda(1-\nu)},$$

since ν by definition is strictly less than one. Similarly, we obtain

$$\begin{aligned}
\mathrm{E}\left[(X_n^*)^2\right] &= \mathrm{E}\left[\alpha\sum_{j=0}^{\infty}\nu^j S_{n-1-j}\alpha\sum_{k=0}^{\infty}\nu^k S_{n-1-k}\right] \\
&= \alpha^2\mathrm{E}\left[\sum_{k=0}^{\infty}\sum_{j=0}^{k}\nu^j S_{n-1-j}\nu^{k-j}S_{n-1-k+j}\right] \\
&= \alpha^2\sum_{k=0}^{\infty}\sum_{j=0}^{k}\nu^k \mathrm{E}\left[S_{n-1-j}S_{n-1-k+j}\right] \\
&= \alpha^2\sum_{k=0}^{\infty}\nu^k \begin{cases} R(0) + 2\sum_{j=1}^{r} R(2j), & k = 2r, \\ 2\sum_{j=1}^{r} R(2j-1), & k = 2r-1. \end{cases}
\end{aligned}$$

Then, we regroup the terms of the last series to get

$$\begin{aligned}
\mathrm{E}\left[(X_n^*)^2\right] &= \alpha^2 R(0)\sum_{j=0}^{\infty}\nu^{2j} + 2\alpha^2\sum_{k=1}^{\infty}R(k)\nu^k\sum_{j=0}^{\infty}\nu^{2j} \\
&= \frac{\alpha^2}{1-\nu^2}\left[R(0) + 2\sum_{k=1}^{\infty}\nu^k R(k)\right].
\end{aligned}$$

■

Remark 5.1 *Often, the covariance function $C(k)$ is used instead of the correlation function $R(k)$. Then, the expression of the second moment becomes*

$$\mathrm{E}\left[(X_n^*)^2\right] = \frac{\alpha^2}{1-\nu^2}\left[V + 2\sum_{k=1}^{\infty}\nu^k C(k)\right] + \frac{\alpha^2}{\lambda^2(1-\nu)^2}.$$

By normalizing with respect to the average inter-loss time, we get

$$\mathrm{E}\left[(X_n^*)^2\right] = \frac{\alpha^2}{\lambda^2(1-\nu^2)}\left[\hat{V} + 2\sum_{k=1}^{\infty}\nu^k \hat{C}(k)\right] + \frac{\alpha^2}{\lambda^2(1-\nu)^2}. \tag{5.5}$$

Remark 5.2 *The expectations computed in (5.3) and (5.4) are taken with respect to loss instants. These expectations are also referred as Palm expectations in the context of point processes [22].*

Remark 5.3 *We note the remarkable insensitivity property, that $\mathrm{E}\left[X_n^*\right]$ does not depend on the correlation of inter-loss times nor on their moments of order greater than one.*

5.2.2 Calculation of the throughput

By using the expression (5.2) and the concept of Palm probability, we proceed for the calculation of TCP throughput, that we recall is given by

$$\bar{X} = \lim_{t\to\infty} \frac{1}{t} \int_0^t X(\tau)\mathrm{d}\tau.$$

Our main result is the following close expression for $\bar{X}$ as a function of the loss intensity λ, the correlation (or covariance) functions, the linear increase factor α (thus b and RTT) and the multiplicative-decrease factor ν. As we will see when validating our results, if all these functions are correctly calculated and are plugged in our explicit expression, we get exactly the throughput of ideal TCP.

Proposition 5.2 *The general expression of TCP throughput is*

$$\bar{X} = \lambda\alpha \left[\frac{1}{2}R(0) + \sum_{k=1}^{\infty} \nu^k R(k)\right]. \tag{5.6}$$

Proof: From [44, Ch. 4], the process $X(t)$ is ergodic since the associated discrete-time process $\{X_n\}$ is also ergodic. The ergodicity of the process $\{X_n\}$ follows from [40, p. 14], (5.2) and the ergodicity of the process $\{S_n\}$ we assumed. Thus, the throughput $\bar{X}$ is almost sure equal to $\mathrm{E}[X(t)]$, the expectation of the rate at an arbitrary time point. To calculate $\mathrm{E}[X(t)]$, one can use the following inversion formula from the Palm theory (see e.g., [22, Ch.1 Sec.4])

$$\mathrm{E}[X(t)] = \lambda \mathrm{E}^0 \left[\int_0^{T_1} X(\tau)\mathrm{d}\tau\right], \tag{5.7}$$

where $\mathrm{E}^0[\cdot]$ is an expectation associated with Palm distribution. In particular, $P^0\{T_0 = 0\} = 1$. We recall that Palm distribution is no other than the distribution of the rate upon loss moments in the stationary regime. The term between brackets on the right-hand side of (5.7) is no other than the area located between the process $X(t)$, the time axis, and the two instants T_0 and T_1, given that $T_0 = 0$ and the process $X(t)$ is in its stationary regime at time 0. Using formula (5.7) and expression (5.2), we can write

$$\begin{aligned}
\bar{X} = \mathrm{E}[X(t)] &= \lambda \mathrm{E}^0 \left[\int_0^{T_1} (\nu X_0 + \alpha\tau)\mathrm{d}\tau\right] = \lambda \mathrm{E}^0 \left[\nu X_0 S_0 + \frac{\alpha}{2} S_0^2\right] \\
&= \lambda \mathrm{E}^0 \left[\alpha\nu \sum_{k=0}^{\infty} \nu^k S_{-1-k} S_0\right] + \frac{\lambda\alpha}{2} \mathrm{E}^0\left[S_0^2\right] \\
&= \lambda\alpha \sum_{k=0}^{\infty} \nu^{k+1} R(k+1) + \frac{\lambda\alpha}{2} R(0) \\
&= \lambda\alpha \left[\frac{1}{2}R(0) + \sum_{k=1}^{\infty} \nu^k R(k)\right]
\end{aligned}$$

■

Remark 5.4 *Note that formula (5.6) can be also rewritten in terms of the second moment of the transmission rate at loss instants,*

$$\bar{X} = \frac{\lambda(1-\nu^2)}{2\alpha}\mathrm{E}\left[(X_n^*)^2\right].$$

The previous remark is very important since it tells us exactly where the previous works on TCP modeling fail in their calculation of the throughput (e.g., [92, 105]). This remark says that the calculation of the second moment of X_n^* is unavoidable for an exact calculation of the throughput. However, this second moment cannot be simply calculated since as we see in (5.4), it is a function of all the functions of the loss process not only its intensity. To avoid this hard calculation, the previous works on TCP make the simplistic assumption that the second moment of X_n^* is equal to the square of its first moment and derive the expression of the throughput as a function of the intensity of losses. Note that the first moment of X_n^* can be directly calculated from the stochastic difference equation (5.2) and it does not depend on the correlation of losses nor on their moments of order greater than one (Equation (5.3)). We know that this simplistic assumption only holds for deterministic loss processes where the window of TCP varies in a periodic and deterministic manner. Hence, we conclude that all works making such simplistic assumption follow the fixed-point approach.

Remark 5.5 *The throughput can be also expressed as a function of the covariance functions $C(k)$,*

$$\bar{X} = \lambda\alpha[\frac{1}{2}V + \sum_{k=1}^{\infty}\nu^k C(k)] + \frac{\alpha(1+\nu)}{2\lambda(1-\nu)}.$$

By normalizing with respect to the average inter-loss time, we get

$$\bar{X} = \frac{\alpha}{\lambda}[\frac{1}{2}\hat{V} + \sum_{k=1}^{\infty}\nu^k \hat{C}(k)] + \frac{\alpha(1+\nu)}{2\lambda(1-\nu)}. \tag{5.8}$$

5.2.3 Generalization of the square root formula

The throughput of TCP has been often calculated as a function of the probability that a TCP packet is lost, or more precisely as a function of the probability that a packet causes the source to reduce its window [84, 92, 105]. Denote this probability by p. Using a fixed-point approach, it has been shown that the throughput of TCP is inversely proportional to RTT and the square root of p. For a loss probability p, an average round-trip time RTT and a reduction factor $\nu = 0.5$, the throughput of TCP in packets/s has been shown to be equal to

$$\bar{X} = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} \tag{5.9}$$

This is the famous square root formula for TCP throughput. In [105], the authors found that this formula is only valid for small loss probabilities (less than 5%) since it does not account for

the timeout mechanism. For high loss probabilities, one must correct the square root formula in the way we presented in Section 3.2.2. Indeed when p increases, the probability that a congestion event results in a timeout (the function $Q(p)$ in Section 3.2.2) increases and the final throughput ($\bar{X}_f$ in Equation (3.1)) starts to differ from the throughput $\bar{X}$ given by the square root formula.

Using our explicit expression of TCP throughput, we generalize in this section the square root formula so that to account for any distribution of inter-loss times and for any correlation of losses. Recall that the square root formula has been established for deterministic losses. Our first and important result is that, even in the most general case, the throughput of TCP is always inversely proportional to RTT and to the square root of p. Our second result is that the throughput of TCP is also a function of all the correlation functions of the loss process. When we introduce p, the intensity of losses disappears and the correlation functions are substituted by the normalized covariance functions. This results in an elegant expression of TCP throughput that explains clearly the impact of the distribution and correlation of inter-loss times on the performance of TCP congestion control and that can be easily specified to any particular loss process.

Denote by $A(t)$ the number of packets transmitted on the TCP connection until time t, and denote by $L(t)$ the number of loss events until time t. The packet drop probability p is simply equal to

$$p = \lim_{t\to\infty} \frac{L(t)}{A(t)} = \lim_{t\to\infty} \frac{\lambda t}{\int_0^t X(\tau)\mathrm{d}\tau} = \frac{\lambda}{\bar{X}} = \frac{1}{s\bar{X}}. \tag{5.10}$$

Note that if a congestion event results in the loss of multiple packets, p will denote the probability that a packet is lost and that it causes the reduction of the congestion window. In [105], packets are assumed to be lost in bursts with a burst of losses causing one reduction of the congestion window and with p being the probability that a packet is the first loss in a burst of losses. Now, it is up to the person to decide on how to calculate p in reality. Some people calculate it as being the ratio of the total number of packets dropped and the total number of packets transmitted. This is the case of works using the square root formula to study the performance of TCP in presence of active queues [48, 58, 98]. Such method for the calculation of p clearly results in an overestimation of p (hence in an underestimation of TCP throughput) when congestion events result in the loss of multiple packets. Other people calculate it in the same way we defined it [63, 105]. We will follow this latter approach for the calculation of p when validating the model. We believe that this is how p must be calculated in presence of new versions of TCP that are able to recover from multiple packet losses in the same round-trip time without reducing multiple times their windows.

The expression of p as a function of λ and $\bar{X}$ (Equation (5.10)) allows us to write our main result in another form so as to grasp the influence of p and RTT on the throughput for a general distribution of inter-loss times. By substituting λ in (5.8) by its value as a function of p and $\bar{X}$,

we get the following general version of the square root formula,

$$\bar{X} = \frac{1}{RTT\sqrt{pb}}\sqrt{\frac{1+\nu}{2(1-\nu)} + \frac{1}{2}\hat{V} + \sum_{k=1}^{\infty}\nu^k\hat{C}(k)}. \tag{5.11}$$

Recall that $\hat{V}$ and $\hat{C}(k)$ are respectively the variance and covariance functions of inter-loss times normalized to the square of the average inter-loss time. The intensity of losses disappears and all what we need to do is to calculate the functions $\hat{V}$ and $\hat{C}(k)$ in order to get the throughput for a certain loss process. Note also how we can get the original square root formula (Equation (5.9)) by taking a deterministic loss process, this means by taking the variance and covariance functions equal to zero.

5.2.4 Loss process functions and TCP performance

Three terms figure under the root in the general square root formula (Equation (5.11)). The first term from the left corresponds to the intensity of losses. It is the only term considered by the previous models for TCP [92, 105]. If considered alone, this means that we are assuming that the loss process is deterministic and what we get is the original square root formula (5.9).

The second term represents the variation of inter-loss times. It says clearly that the throughput of TCP increases with the variance of times between losses. This is in parallel to our result in Chapter 4) which says that the throughput of TCP increases when losses tend to appear in bursts. This second term tells us that the variance of inter-loss times should be correctly estimated for a correct calculation of TCP throughput: underestimating the variance leads to an underestimation of TCP throughput and overestimating it leads to an overestimation. For example, assuming that losses are deterministic when they are Poisson results in an underestimation of the throughput. In fact, what we need to calculate is not the variance but the ratio of the variance and the square of the average inter-loss time ($\hat{V}$). This ratio indicates the type of the distribution of inter-loss times. For example, it is equal to 0 for deterministic losses and to 1 for Poisson losses whatever is the intensity of losses. Note here that all this reasoning holds if our linear model for TCP window evolution is correct. As we will see in the section on validation of the results, this is not always the case. The error introduced by a wrong choice of the loss process can be canceled by the error introduced by a wrong model for TCP. It is here that figures the advantage of our technique for model validation we introduced in Chapter 3.

Now, the third and last term under the root represents the correlation of losses. It is equal to zero if the loss process exhibits no correlation, this means if inter-loss times are independent and identically distributed. A covariance function $\hat{C}(k)$ takes its value between $-\hat{V}$ and $\hat{V}$. This value depends on whether the loss process is positively or negatively correlated and on the importance of the correlation. It is clear that due to the geometrical decrease in the weights of $\hat{C}(k)$, a small number of covariance functions are sufficient to predict the throughput of TCP

even if the loss process is highly correlated. Indeed, due to the multiplicative-decrease factor ν, the window evolution becomes independent of the past after a small number of loss events.

5.2.5 Examples of loss processes

Let us specify our results to some particular loss processes. We already saw that the original square root formula is a particular case of our general result.

IID losses (General Renewal Process)

First, we model the loss process as a general renewal process. Namely, we assume that $\{S_n\}$ are independent and identically distributed random variables. Thus, except for $k = 0$, the covariance coefficients are equal to zero. The formulas (5.3), (5.5), (5.8) and (5.11) take the following form.

Proposition 5.3 *Let $\{S_n\}$ be i.i.d.. Then,*

$$\mathrm{E}\left[X_n^*\right] = \frac{\alpha}{\lambda(1-\nu)},$$

$$\mathrm{E}\left[(X_n^*)^2\right] = \frac{\alpha^2}{\lambda^2(1-\nu)}\left[\frac{1}{1-\nu} + \frac{\hat{V}}{1+\nu}\right],$$

$$\bar{X} = \frac{\alpha}{2\lambda}\left[\frac{1+\nu}{1-\nu} + \hat{V}\right] = \frac{1}{RTT\sqrt{pb}}\sqrt{\frac{1+\nu}{2(1-\nu)} + \frac{1}{2}\hat{V}}. \tag{5.12}$$

In particular, if the inter-loss times are exponentially distributed, we have

$$\bar{X} = \frac{\alpha}{\lambda(1-\nu)} = \frac{1}{RTT\sqrt{pb}}\sqrt{\frac{1}{1-\nu}}. \tag{5.13}$$

For $\nu = 0.5$, we get the same expression for TCP throughput as that obtained in [96]. If the inter-loss times are deterministic, we get

$$\bar{X} = \frac{\alpha}{2\lambda}\frac{1+\nu}{1-\nu} = \frac{1}{RTT\sqrt{pb}}\sqrt{\frac{1+\nu}{2(1-\nu)}}$$

For $\nu = 0.5$, this gives the original square root formula (Equation (5.9)). If one applies our corrections (3.1) and (3.2) to the original square root formula, he will find a similar result to that obtained by the detailed packet-level model in [105]. In fact, [105] considers (indirectly) a deterministic model for inter-loss times and this is simply due to its assumption that processes $\{X_n^*\}$ and $\{S_n\}$ are mutually independent. These two processes are mutually independent in the only case of deterministic times between loss events. Later, we will show with measurements the similarity between the model in [105] and our model with deterministic losses.

Correlated losses modeled as a Markovian Arrival Process

In this section we consider correlated losses which are modeled by a Markovian Arrival Process (MAP) [20, 79, 100]. An arrival (or the occurrence of a point) in MAP corresponds to the occurrence of a congestion event in our context. It was shown in [20] that for a given general point process, there is a sequence of MAPs which converges to the point process in distribution. In particular, this implies that in principle any point process can be approximated by appropriate MAPs. Furthermore, PH-renewal process [100] and Markov Modulated Poisson Process (MMPP) [59] are particular cases of the Markovian Arrival Process.

Let us briefly review the definition and some properties of MAP. Similarly to our Markovian model for losses in Chapter 4, a MAP process consists in a Poisson process that changes its rate according to a multi-state Markov chain. The transitions of the underlying Markov chain may happen at the moments of arrivals or between arrivals. Let $N(t)$ be a counting process associated with MAP, that is $N(t)$ is the number of arrivals in the interval $(0,t]$. Also, let $J(t)$ be an auxiliary variable that indicates the state of the underlying Markov chain. Consider that this Markov chain has m states denoted $1,\dots,m$. Then, MAP can be described in terms of a two-dimensional Markov process $\{N(t),J(t)\}$ on the state space $\{(i,j)|i\geq 0, 1\leq j\leq m\}$ with the following infinitesimal generator [80]

$$Q=\begin{bmatrix} C & D & 0 & 0 & \cdots \\ 0 & C & D & 0 & \cdots \\ 0 & 0 & C & D & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

where the matrix $C\in\mathbf{R}^{m\times m}$ governs the transitions of the process J without arrivals and it has negative diagonal elements and nonnegative off-diagonal elements. The matrix $D\in\mathbf{R}^{m\times m}$ governs the transitions of J with the simultaneous arrivals and it has nonnegative elements. Thus, the underlying Markov process $J(t)$ has the following infinitesimal generator,

$$\bar{Q}=C+D.$$

Furthermore, we assume that $\bar{Q}\neq C$ and C is a stable matrix. This ensures that $J(t)$ does not get absorbed in a class of states in which arrivals stop. When $J(t)=i$, the rate of transitions to state $j\neq i$ is $\bar{Q}_{ij}$. If such a transition occurs then an arrival occurs simultaneously with the transition with probability $D_{ij}/(-C_{ii}-D_{ii})$. Note that MAP becomes MMPP with infinitesimal generator R and arrival rate matrix Λ, if we take $C=R-\Lambda$ and $D=\Lambda$. The difference between MMPP and MAP is simply that with MMPP arrivals and transitions cannot appear simultaneously, however this is possible with MAP.

Let $\{S_n\}$ be the sequence of inter-arrival times for MAP and let $\{J_n\}$ be the sequence of the states of the underlying Markov process at the arrival epochs. Then, $\{J_n,S_n\}$ is a Markov

renewal process [73] with the following transition probability matrix [100],

$$F(x) = \left(\int_0^x \exp\{Cu\} \mathrm{d}u \right) D = (I - \exp\{Cx\})(-C)^{-1} D.$$

$F_{ij}(x)$ represents the probability that the time until the next arrival is less than x given that the underlying Markov chain is currently in state i and will be in state j upon the next arrival. $T = F(\infty) = -C^{-1}D$ is a transition probability matrix of a discrete time Markov chain embedded at the instants of arrivals (process J_n). Let μ be the stationary distribution this discrete time Markov chain. If we take the initial distribution of the underlying Markov chain $J(t)$ as μ, the arrival process becomes interval-stationary or event-stationary. The event-stationary version of MAP has the following joint distribution function for the inter-arrival times [79],

$$F_{S_0 \cdots S_n}(x_0, ..., x_n) = \mu \prod_{i=0}^{n} \{(I - \exp\{Cx_i\})T\} e. \tag{5.14}$$

e denotes the column vector of ones,

$$e = (1, \ldots, 1)^T.$$

Consequently, the joint Laplace Stieltjes Transform is given by,

$$f(z_0, ..., z_n) = \mathrm{E}\left[\exp\{-\sum_{k=0}^{n} z_k S_k\} \right] = \mu \prod_{k=0}^{n} \{(z_k I - C)^{-1} D\} e. \tag{5.15}$$

Next, using the Laplace Stieltjes transform (5.15), we can easily calculate the first two moments and the correlation functions of the inter-arrival time process. Namely,

$$\mathrm{E}[S_n] = -\frac{d}{dz}(\mu(zI - C)^{-1} De)|_{z=0} = -\mu C^{-1} e, \tag{5.16}$$

$$R(0) = \mathrm{E}\left[S_n^2\right] = \frac{d^2}{dz^2}(\mu(zI - C)^{-1} De)|_{z=0} = 2\mu C^{-2} e, \tag{5.17}$$

$$\begin{aligned} R(k) = \mathrm{E}[S_n S_{n-k}] &= \frac{\partial^2}{\partial z_0 \partial z_k} f(z_0, ..., z_k)|_{z_i=0} \\ &= \mu C^{-2} D T^{k-1} C^{-2} De. \end{aligned} \tag{5.18}$$

To derive the above expressions, we have used the following formula for the differentiation of an inverse matrix-valued function: $(A^{-1}(z))' = -A^{-1}(z)A'(z)A^{-1}(z)$ [36]. We have also used the relation: $-C^{-1}De = Te = e$. Now, employing (5.3), (5.4), (5.16), (5.17) and (5.18), we can calculate the first two moments of the process X_n^*.

Proposition 5.4 *Let the loss process $\{S_n\}$ be represented by MAP. Then,*

$$\mathrm{E}[X_n^*] = -\frac{\alpha}{1-\nu} \mu C^{-1} e,$$

$$\mathrm{E}\left[(X_n^*)^2\right] = \frac{2\alpha^2}{1-\nu^2} \mu \left(C^{-2} + \nu C^{-2} D [I - \nu T]^{-1} C^{-2} D \right) e.$$

Proof: The above formulas are immediately obtained from (5.3) and (5.4) with the help of the following derivation,

$$\begin{aligned}\sum_{k=1}^{\infty}\nu^k R(k) &= \mu C^{-2}D\sum_{k=1}^{\infty}\nu^k T^{k-1}C^{-2}De \\ &= \mu C^{-2}D\nu\sum_{k=0}^{\infty}\nu^k T^k C^{-2}De = \nu\mu C^{-2}D[I-\nu T]^{-1}C^{-2}De.\end{aligned}$$

■

Next, using (5.6), we calculate the throughput of TCP when the loss process is modeled by MAP.

Proposition 5.5 *Let $\{S_n\}$ be a Markovian Arrival Process. Then, the throughput of TCP is given by,*

$$\bar{X} = -\frac{\alpha}{\mu C^{-1}e}\mu(C^{-2}+\nu C^{-2}D[I-\nu T]^{-1}C^{-2}D)e. \tag{5.19}$$

5.2.6 Bounds for the model with window limitation

Our previous expressions of the throughput don't work when the window advertised by the receiver is reached (Figure 5.2). If it is the case, the model becomes sub-linear and an exact calculation of the throughput seems to be impossible except for some particular loss processes as deterministic and Poisson. The case of a Poisson loss process will be addressed in the next chapter. We will see that even for a simple Poisson process, the calculation of the throughput is quite difficult. Now for a more general loss process, some approximation of the throughput can be always found. One of the possible approximations is to use a fixed-point approach and to assume that the rate varies in the stationary regime in a deterministic manner between $\nu\mathrm{E}[X_n^*]$ and $\mathrm{E}[X_n^*]$, of course when the maximum limit M is frequently reached. See Equation (5.3) for the expression of $\mathrm{E}[X_n^*]$. One can use our previous expression of the throughput (5.8) whenever $\mathrm{E}[X_n^*]$ is less than M. Once $\mathrm{E}[X_n^*]$ becomes larger than M, the throughput can be approximated using the fixed-point approach (Figure 5.3). For a loss process of intensity λ, this approximation gives,

$$\bar{X} \simeq M - \frac{\lambda M^2(1-\nu^2)}{2\alpha}. \tag{5.20}$$

This is the kind of approximation used in [105]. In our Markovian model we used such an approach (Section 4.4) but we profited from the existence of the Markov chain for the path to derive more precise approximation of the throughput [11]. One of the drawbacks of the fixed-point approximation is that we have more than one expression for the throughput and we have to switch between them based on some conditions on the moments of the process X_n^*. This may result in a jump in the throughput when we switch from one expression to another since the two expressions may not be equal at the boundary. Consider for example the case of a Poisson loss process. Suppose that we condition on $\mathrm{E}[X_n^*]$ (Equation (5.3)) to decide on whether to

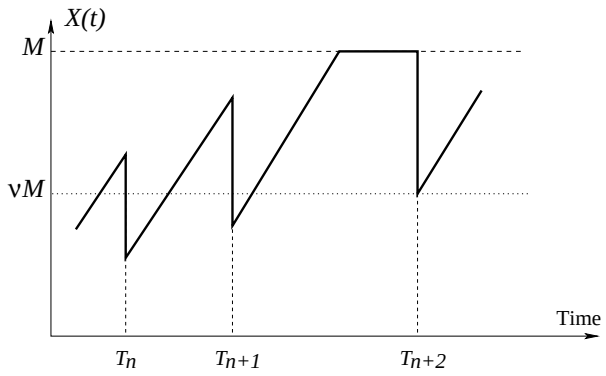


Figure 5.2: Rate evolution with limitation

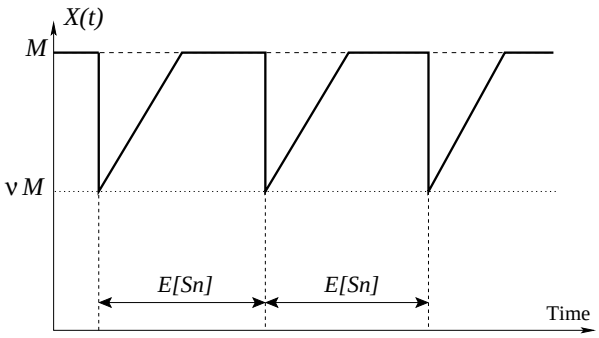


Figure 5.3: Fixed-point approach

approximate the throughput with (5.12) or with (5.20). It is clear that when $\mathrm{E}[X_n^*] = M$, these two expressions of the throughput are not equal. They are however equal when the loss process is deterministic. One possible solution to this jump is to always take the minimum throughput given by the different expressions instead of switching between them based on some conditions on the moments of the process X_n^*. For example, in the case of Poisson losses, we can write the throughput as

$$\bar{X} = \min\left(\frac{\alpha}{\lambda(1-\nu)}, M - \frac{\lambda M^2(1-\nu)^2)}{2\alpha}\right),$$

instead of

$$\bar{X} = \begin{cases} \frac{\alpha}{\lambda(1-\nu)} & \text{if } \mathrm{E}[X_n^*] = \frac{\alpha}{\lambda(1-\nu)} < M \\ M - \frac{\lambda M^2(1-\nu)^2)}{2\alpha} & \text{otherwise} \end{cases}$$

In this chapter we follow a different approach for the approximation of the throughput when the receiver window is reached. We shall calculate bounds for the throughput which are at the same time good approximations. The advantage of our bounds is that they are valid for any receiver window. They converge to (5.6) when the receiver window is large and they approximate the throughput when the receiver window is small. Thus, we no longer need to condition on the moments of the process X_n^* in order to switch from one expression of the throughput to another.

Let us proceed for the calculation of the bounds. When the problem of window limitation exists, the stochastic difference Equation (5.1) is modified to the following form

$$X_{n+1} = M \wedge (\nu X_n + \alpha S_n), \tag{5.21}$$

where $\wedge$ stands for the *minimum* operation. It is clear how the model becomes nonlinear and how the solution for the moments of the rate is difficult. This nonlinear stochastic difference equation can be rewritten as,

$$X_{n+1} = \nu X_n + \alpha S_n \wedge (M - \nu X_n).$$

Since $0 \leq X_n \leq M$, we have

$$\nu X_n + \alpha S_n \wedge (1-\nu)M \leq X_{n+1} \leq \nu X_n + \alpha S_n \wedge M.$$

The above inequalities prompt us to derive lower and upper bounds for the throughput using the next auxiliary stochastic processes defined on the same probability space as X_n,

$$\check{X}_{n+1} = \nu\check{X}_n + (\bar{\nu}M) \wedge (\alpha S_n) = \nu\check{X}_n + \alpha(\frac{\bar{\nu}M}{\alpha} \wedge S_n) = \nu\check{X}_n + \alpha\check{S}_n, \tag{5.22}$$

and

$$\hat{X}_{n+1} = \nu\hat{X}_n + M \wedge (\alpha S_n) = \nu\hat{X}_n + \alpha(\frac{M}{\alpha} \wedge S_n) = \nu\hat{X}_n + \alpha\hat{S}_n. \tag{5.23}$$

with $\bar{\nu} = 1 - \nu$, $\check{S}_n = \frac{\bar{\nu}M}{\alpha} \wedge S_n$ and $\hat{S}_n = \frac{M}{\alpha} \wedge S_n$.

The processes $\check{X}_n$ and $\hat{X}_n$ are governed by linear stochastic difference equations similar to the one we studied in the case of no limitation on the congestion window (Equation (5.1)). Our previous results can be then used for the calculation of their moments, of course after the substitution of the functions of process S_n by those of processes $\check{S}_n$ and $\hat{S}_n$. We also deduce that the processes $\check{X}_n$ and $\hat{X}_n$ converge to a unique stationary regime whatever are their initial states. But before deriving the bounds, let us prove that the rate process X_n still converges to a unique stationary regime for an arbitrary initial state. We use a Loynes-type construction for this purpose. From now until the end of this section, we consider that the rate process X_n is governed by the nonlinear stochastic difference Equation (5.21). The processes $\check{X}_n$ and $\hat{X}_n$ are respectively governed by the linear stochastic difference equations (5.22) and (5.23).

Theorem 5.1 *Assume that $\{S_n\}$ is a stationary process. Then, there exists a stationary process $\{X_n^*\}$ defined on the same probability space and satisfying the recursion (5.21). Furthermore, for any initial state X_0, we have P-a.s.*

$$\lim_{n\to\infty} \sup_{k\geq n} |X_k - X_k^*| = 0. \tag{5.24}$$

That is, for any initial state X_0, the process $\{X_k\}_{k\geq n}$ converges in distribution to the stationary process $\{X_n^*\}$ as $n \to \infty$.

Proof: Define on the same probability space the family of processes $\{X_k^{(n)}, k \in Z\}$, $n = 0, 1, ...$ as follows. $X_k^{(n)} = 0$ for $k \leq -n$, and for $k > -n$ it is given by the recursion (5.21). For each k, $X_k^{(n)}$ is increasing with respect to n and thus it has a limit (obviously finite) which we denote by X_k^*. This limit satisfies (5.21) since for every n, $X_k^{(n)}$ satisfies it. Finally, the stationarity of the sequence $\{S_n\}$ implies that $\{X_n^*\}$ is stationary as well. The convergence of X_n to X_n^* follows from,

$$|X_{n+1} - X_{n+1}^*| = |M \wedge (\nu X_n + \alpha S_n) - M \wedge (\nu X_n^* + \alpha S_n)| \leq \nu |X_n - X_n^*|.$$

To show that the above inequality holds, one needs to consider four cases. If the both values of $(\nu X_n + \alpha S_n)$ and $(\nu X_n^* + \alpha S_n)$ are less or alternatively greater than M, then the inequality is obvious. Let us consider not so obvious cases. For example, let $(\nu X_n + \alpha S_n) > M$ and $(\nu X_n^* + \alpha S_n) < M$, then

$$\begin{aligned} |X_{n+1} - X_{n+1}^*| &= M - (\nu X_n^* + \alpha S_n) \\ &\leq (\nu X_n + \alpha S_n) - (\nu X_n^* + \alpha S_n) \leq \nu |X_n - X_n^*|. \end{aligned}$$

Thus,

$$|X_n - X_n^*| \leq \nu^n |X_0 - X_0^*| \to 0 \quad \text{as} \quad n \to \infty.$$

Since both X_0 and X_0^* are finite (they are bounded between 0 and M), this implies (6.2) which concludes the proof. ■

Now, we use the two processes $\{\check{X}_n\}$ and $\{\hat{X}_n\}$ to find bounds for the throughput.

Proposition 5.6 *Let $\{S_n\}$ be a stationary stochastic point process. Then for all $n \geq 0$, we have*

$$\check{X}_n^* \leq X_n^* \leq \hat{X}_n^*.$$

Proof: Consider the proof for the lower bound. Without loss of generality, we assume that $X_0 = \check{X}_0$. We show by induction that for all $n \geq 0$, $\check{X}_n \leq X_n$. This holds for $n = 0$. Assume that it holds for $n = k$. Then, consider two cases $S_k \leq \frac{\bar{\nu}M}{\alpha}$ and $S_k > \frac{\bar{\nu}M}{\alpha}$. For $S_k \leq \frac{\bar{\nu}M}{\alpha}$, one has

$$X_{k+1} = \nu X_k + \alpha S_k \geq \nu \check{X}_k + \alpha S_k = \check{X}_{k+1}.$$

And if $S_k \geq \frac{\bar{\nu}M}{\alpha}$, then

$$\begin{aligned} X_{k+1} &= \nu X_k + (M - \nu X_k) \wedge (\alpha S_k) \\ &\geq \nu X_k + (M - \nu X_k) \wedge (\bar{\nu} M) = \nu X_k + \bar{\nu} M \geq \nu \check{X}_k + \bar{\nu} M = \check{X}_{k+1}. \end{aligned}$$

The first inequality is true, since $X_k \leq M$. Hence, $\check{X}_{k+1} \leq X_{k+1}$ and according to the induction principle, the inequality $\check{X}_n \leq X_n$ holds for all $n \geq 0$. Since by the results of [67] and our Theorem 5.1 both processes $\{\check{X}_n\}$ and $\{X_n\}$ converge to the stationary regime for any initial state, we can let n go to infinity. This results in the lower bound. The upper bound is obtained in the similar manner by using the auxiliary process $\{\hat{X}_n\}$ defined in (5.23). ■

Proposition 5.7 *Let $\check{s} = \mathrm{E}\left[\check{S}_n\right]$, $\check{s}^{(2)} = \mathrm{E}\left[(\check{S}_n)^2\right]$, $\check{R}(k) = \mathrm{E}\left[S_n \check{S}_{n-k}\right]$ and $\hat{s} = \mathrm{E}\left[\hat{S}_n\right]$, $\hat{s}^{(2)} = \mathrm{E}\left[(\hat{S}_n)^2\right]$, $\hat{R}(k) = \mathrm{E}\left[S_n \hat{S}_{n-k}\right]$. Then, the lower and upper bounds for the throughput are given by*

$$\bar{X} \geq \alpha\lambda \left(\check{R}(0) - \frac{1}{2}\check{s}^{(2)} + \sum_{k=1}^{\infty} \nu^k \check{R}(k) \right), \tag{5.25}$$

$$\bar{X} \leq \alpha\lambda \left(\hat{R}(0) - \frac{1}{2}\hat{s}^{(2)} + \sum_{k=1}^{\infty} \nu^k \hat{R}(k) \right). \tag{5.26}$$

Proof: To obtain the lower bound for the throughput, we again use the auxiliary process $\{\check{X}_n\}$ defined in (5.22). Suppose that $\{\check{X}_n\}$ is in its stationary regime and define

$$\check{X}(t) = \begin{cases} \nu \check{X}_n^* + \alpha t, & t \in [T_n, \check{T}_n], \\ \nu \check{X}_n^* + \alpha \check{S}_n, & t \in [\check{T}_n, T_{n+1}], \end{cases} \tag{5.27}$$

where $\check{T}_n = T_n + \check{S}_n$. See Figure 5.4 for how $\check{X}(t)$ is defined. Similarly to (5.2), one can write the expression of the stationary version of $\check{X}_n$,

$$\check{X}_n^* = \alpha \sum_{k=0}^{\infty} \nu^k \check{S}_{n-1-k}. \tag{5.28}$$

Now, using (5.27), (5.28), and the inversion formula (5.7), we obtain the expression of the lower bound

$$\begin{aligned}
\bar{X} &= \lambda \mathrm{E}^0 \left[\int_0^{T_1} X(\tau) \mathrm{d}\tau \right] \geq \lambda \mathrm{E}^0 \left[\int_0^{T_1} \check{X}(\tau) \mathrm{d}\tau \right] \\
&= \lambda \mathrm{E}^0 \left[\int_0^{\check{S}_0} (\nu \check{X}_0^* + \alpha\tau) \mathrm{d}\tau + \int_{\check{S}_0}^{S_0} (\nu \check{X}_0^* + \alpha \check{S}_0) \mathrm{d}\tau \right] \\
&= \lambda \mathrm{E}^0 \left[\nu \check{X}_0^* \check{S}_0 + \frac{\alpha}{2} \check{S}_0^2 + (\nu \check{X}_0^* + \alpha \check{S}_0)(S_0 - \check{S}_0) \right] \\
&= \lambda \mathrm{E}^0 \left[\nu \check{X}_0^* S_0 + \alpha \check{S}_0 S_0 - \frac{\alpha}{2} \check{S}_0^2 \right] \\
&= \lambda \mathrm{E}^0 \left[\nu\alpha \sum_{k=0}^{\infty} \nu^k \check{S}_{-1-k} S_0 + \alpha \check{S}_0 S_0 - \frac{\alpha}{2} \check{S}_0^2 \right] \\
&= \alpha\lambda \left(\sum_{k=1}^{\infty} \nu^k \check{R}(k) + \check{R}(0) - \frac{1}{2} \check{s}^{(2)} \right).
\end{aligned}$$

Now, by using the auxiliary process $\{\hat{X}_n\}$ defined in (5.23), one can calculate the upper bound for the throughput in a similar way. ■

Note that the two bounds given in Proposition 5.7 coincide with the throughput given by (5.6) when $M/(\alpha s) \to \infty$. This is because the two processes $\{\check{S}_n\}$ and $\{\hat{S}_n\}$ coincide with the process $\{S_n\}$ in this case. Thus, our two bounds can be used as an approximation of the throughput in the case of a large receiver window. Now, when $M/(\alpha s) \to 0$ (case of a small receiver window), the upper bound provided in (5.26) goes to $M/(1-\nu)$ which is not possible since we know that the throughput cannot exceed M. We propose to take as an upper bound the minimum of M and the expression in (5.26). The lower bound however converges to the approximation of the throughput in (5.20). One can use our lower bound as an approximation of the throughput for all receiver windows.

Bounds for Poisson and IID loss processes

Assume first that the loss process is Poisson. Then, formulas (5.25) and (5.26) give the following bounds for the throughput,

$$\frac{\alpha}{\lambda(1-\nu)} \left(1 - e^{-\bar{\nu} M \lambda/\alpha} \right) \leq \bar{X} \leq \frac{\alpha}{\lambda(1-\nu)} \left(1 - e^{-M\lambda/\alpha} \right).$$

Note that $\alpha/\lambda(1-\nu)$ is the throughput of TCP when the loss process is Poisson and when there is no maximum limit on the congestion window (Equation (5.13)).

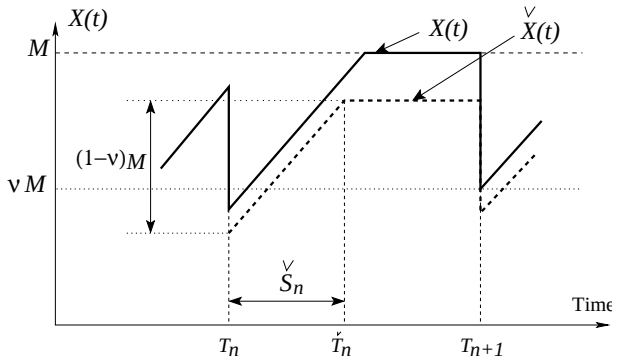


Figure 5.4: A lower bound for the transmission rate

Consider next the more general case of an i.i.d. loss process. The correlation functions $\check{R}(k)$ and $\hat{R}(k)$, for $k \geq 1$, are respectively equal to $s\check{s}$ and $s\hat{s}$. Hence, the bounds for the throughput can be written as,

$$\alpha\lambda\left(\check{R}(0) - \frac{1}{2}\check{s}^{(2)} + \frac{\nu}{\bar{\nu}}s\check{s}\right) \leq \bar{X} \leq \alpha\lambda\left(\hat{R}(0) - \frac{1}{2}\hat{s}^{(2)} + \frac{\nu}{\bar{\nu}}s\hat{s}\right).$$

Bounds for MAP loss process

The functions $\check{s}^{(2)}$, $\hat{s}^{(2)}$, $\check{R}(k)$ and $\hat{R}(k)$ that figure in the expressions of bounds, can be easily calculated for a loss process modeled by a Markovian Arrival Process. Look at Section 5.2.5 for an overview on MAPs. We write,

$$\begin{aligned}
\check{s}^{(2)} = \mathrm{E}\left[\check{S}_n^2\right] &= \int_0^\infty (\bar{\nu}M/\alpha \wedge x)^2 \mu \mathrm{d}F(x)e \\
&= \mu\int_0^\infty (\bar{\nu}M/\alpha \wedge x)^2 \exp\{xC\}\mathrm{d}xDe \\
&= \mu\left(-\frac{2\bar{\nu}M}{\alpha}\exp\left\{\frac{\bar{\nu}M}{\alpha}C\right\}C^{-2}D + 2\left[\exp\left\{\frac{\bar{\nu}M}{\alpha}C\right\} - I\right]C^{-3}D\right)e.
\end{aligned}$$

Similarly, we find

$$\hat{s}^{(2)} = \mathrm{E}\left[\hat{S}_n^2\right] = \mu\left(-2\frac{M}{\alpha}\exp\left\{\frac{M}{\alpha}C\right\}C^{-2}D + 2\left[\exp\left\{\frac{M}{\alpha}C\right\} - I\right]C^{-3}D\right)e.$$

To compute $\check{R}(k)$ and $\check{R}(k)$, $k \geq 1$, we use the joint distribution (5.14).

$$\begin{aligned}
\check{R}(k) &= \int_0^\infty \cdots \int_0^\infty (\bar{\nu}M/\alpha \wedge x_0)x_k \, \mu\exp\{Cx_0\}D\cdots\exp\{Cx_k\}De \, \mathrm{d}x_0\cdots\mathrm{d}x_k \\
&= \mu\left[I - \exp\left\{\frac{\bar{\nu}M}{\alpha}C\right\}\right]C^{-2}DT^{k-1}C^{-2}De,
\end{aligned}$$

and similarly,

$$\hat{R}(k) = \mu\left[I - \exp\left\{\frac{M}{\alpha}C\right\}\right]C^{-2}DT^{k-1}C^{-2}De.$$

For $k = 0$, we get

$$\begin{aligned}
\check{R}(0) &= \mu \int_0^{\infty} (\bar{\nu}M/\alpha \wedge x) x\mu \exp\{Cx\} \mathrm{d}x De \\
&= \mu \left(-\frac{\bar{\nu}M}{\alpha} \exp\left\{ \frac{\bar{\nu}M}{\alpha} C \right\} C^{-2} D + 2 \left[\exp\left\{ \frac{\bar{\nu}M}{\alpha} C \right\} - I \right] C^{-3} D \right) e, \\
\hat{R}(0) &= \mu \left(-\frac{M}{\alpha} \exp\left\{ \frac{M}{\alpha} C \right\} C^{-2} D + 2 \left[\exp\left\{ \frac{M}{\alpha} C \right\} - I \right] C^{-3} D \right) e.
\end{aligned}$$

5.3 Model validation

Using the technique we introduced in Section 3.4, we validate our general model for TCP. The three connections (LAN, MAN, WAN) described in Section 3.1 are considered for this purpose. First, we consider the working hours. We noticed that during these hours the receiver window is not reached and hence the expressions of the throughput we found in this chapter in case of no limitation of TCP rate can be used. The reason for that the receiver window is not reached is an important exogenous traffic at these hours of the day. We compare the result of our general model under different assumptions on the loss process to the throughput of ideal TCP or the so-called exact fluid model. Recall that the throughput of the exact fluid model is calculated numerically after the reconstruction of the rate evolution of ideal TCP from the trace of the connection. The rate evolution of ideal TCP is represented by the straight line in Figure 3.4. Ideal TCP is assumed to increase its rate linearly with time at a rate $\alpha = 1/(bRTT^2)$ and to decrease it by a multiplicative factor $\nu = 0.5$ upon congestion. This comparison tells us how well our model for losses approximates the real loss process. It also tells us how well our calculation is. Normally, if we calculate correctly all the functions of the loss process and we plug them in our expression for the throughput (5.6), we must get the throughput of ideal TCP calculated numerically. In fact, the ideal throughput is exactly the throughput we are trying to estimate in our analysis and which we (and other authors using linear models to study TCP) are claiming represents real TCP throughput. Once we decide on the correctness of our choice of the loss process and the correctness of our approximation of the ideal throughput, we pass to the validation of our model for TCP rate evolution. This is simply done by comparing the ideal throughput to the real one. The ideal throughput is corrected with (3.1) and (3.2) in order to account for timeouts and the packet nature of TCP. The real throughput of a connection is calculated as the ratio of the total number of packets transmitted and the total time of the measurement. We expect the ideal throughput to be close to the real throughput if our model for TCP rate evolution is correct. We will see that it is not always the case and this is mainly due to the sub-linearity of rate increase in some situations. Again, we will highlight the problem of error cancellation we discussed in Section 3.4. The error introduced by a wrong choice of the loss process may be canceled by the error introduced by a wrong modeling of TCP rate evolution. The result will be a low total error and an overestimation of the capacity of the model. The problem of error

cancellation, that we recall is not possible to detect without our technique for separating the validation, leads to wrong conclusions on the type of the loss process seen by the connection and on the evolution of the congestion window. We provide then a comparison of our model to the detailed packet-level model in [105]. We will show in particular that the throughput of this latter model coincides with the throughput of our model under the assumption that inter-loss times are deterministic. In fact, our model with deterministic inter-loss times coincides with that in [105]. Without the corrections we introduced for timeouts and the packet nature of TCP and in the presence of deterministic losses, our model coincides with the original square root formula [92]. Finally, we validate our upper and lower bounds for TCP throughput. The non-working hours on the WAN connection are considered for this purpose.

5.3.1 Validation of the model for losses

We start by studying how well our model predicts the ideal throughput under different assumptions on the loss process. To this end, different loss processes are considered: deterministic, Poisson, general i.i.d., general correlated. The different functions of the loss process seen by the connection (intensity, variance, correlation functions) are calculated from the moments of congestion events detected by our monitoring tool (Section 3.1). Normally, with the general correlated loss process, one must expect that our result will coincide with the ideal throughout. For each loss process type, we calculate the throughput of TCP expected by our model. We use the expression of the throughput we found in the case of no window limitation (5.6), of course specified to the particular distribution of losses. Note that for the general correlated loss process, our expression of the throughput involves an infinite sum of correlation functions. We choose the number of correlation functions so that the result converges. Recall that we are considering working hours which permits to our explicit expression of the throughput to be used.

We plot in Figures 5.5, 5.6, and 5.7 the results for our three connections (LAN, MAN, WAN) as a function of day time. We also plot in the same figures the throughput of the exact fluid model calculated numerically. We clearly notice that our general model gives the same result as the exact fluid model although only five terms (five correlation functions) are considered in the infinite sum in (5.6). The need for only a small number of terms is due to the geometrical decrease in the weights of the correlation functions in the general expression of the throughput (5.6). We also notice that the i.i.d. model gives approximately the same result as the correlated model. Hence, the correlation we saw on our connections is not so important to impact considerably the throughput.

Consider now the Poisson and deterministic cases. We look in Figures 5.5, 5.6 and 5.7 at points where losses are i.i.d.. As we explained in Section 5.2.4, at a constant loss rate, the throughput of TCP increases with the variance of inter-loss times. For this reason, the points for the Poisson case are above those for the deterministic case. On the LAN connection, the loss process is highly bursty (Figures 3.4 and 3.8) and the variance of inter-loss times is larger

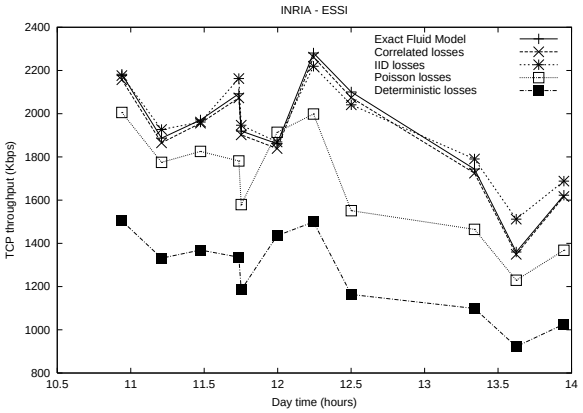


Figure 5.5: LAN: choice of loss process

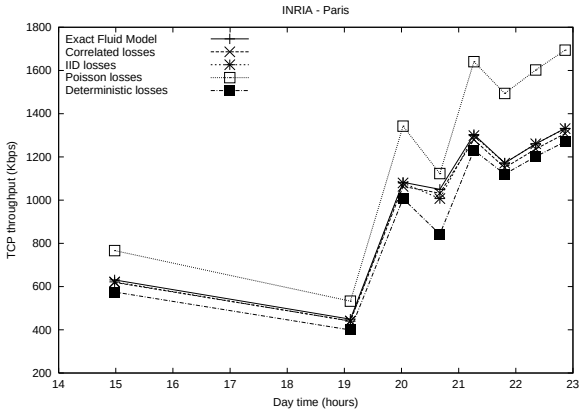


Figure 5.6: MAN: choice of loss process

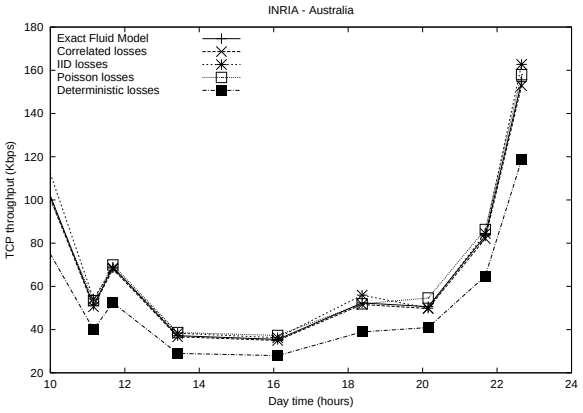


Figure 5.7: WAN: choice of loss process

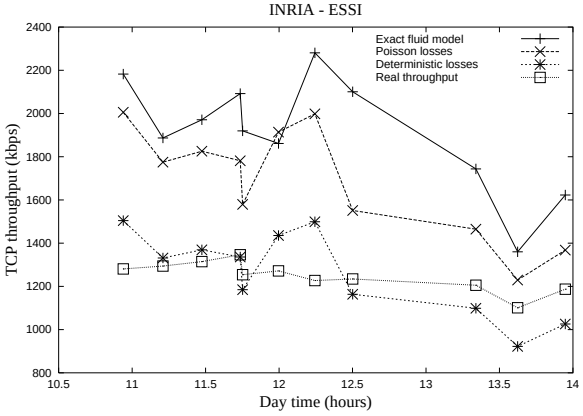


Figure 5.8: LAN: model for TCP

than that of an exponential distribution. Thus, deterministic and Poisson losses lead to an underestimation of the ideal throughput with Poisson losses giving better results. On MAN, the times between losses follow approximately the Normal distribution (Figure 3.10) with a variance smaller than that of an exponential distribution. For this reason, the ideal throughput is located between the throughput in case of deterministic losses and that in case of Poisson losses. Finally on WAN, the loss process is close to Poisson (Figure 3.12) and the ideal throughput is approximately equal to that given by our model for Poisson losses. Assuming that losses are deterministic on WAN underestimates the ideal throughout. These three figures validate our conclusions on the sensitivity of the throughput of TCP to the variance of times between losses. They also validate the correctness of our calculation since with a general loss process we are able to predict correctly the ideal throughput of TCP.

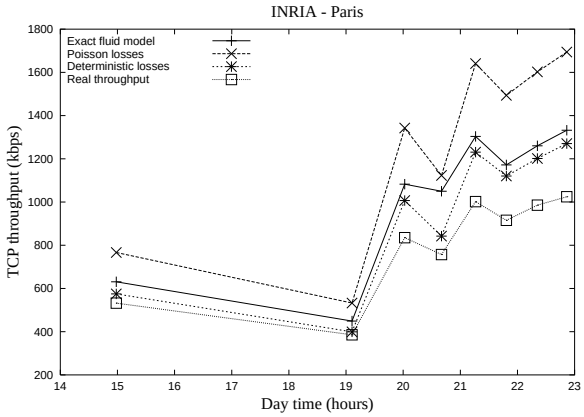


Figure 5.9: MAN: model for TCP

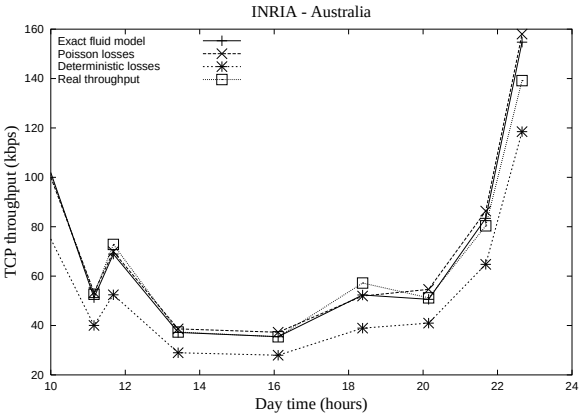


Figure 5.10: WAN: model for TCP

5.3.2 Validation of the model for TCP

In this section we compare the exact fluid model to real TCP on the three connections we are considering. The results are provided in Figures 5.8, 5.9 and 5.10 for the LAN, MAN and WAN connections respectively. In these figures, we plot the real throughput, the ideal the throughput, the result of our model for Poisson losses and the result of our model for deterministic losses. On the LAN and MAN connections, the exact fluid model overestimates real TCP. The overestimation increases with the real throughput. As we explained in Section 3.2.1, this overestimation is mainly due to the sub-linearity of TCP congestion window evolution which in turn is due to the correlation of window and round-trip time. Using the correct functions of the loss process does not yield the best approximation of the real throughout since we still have the error introduced by our linear model for rate evolution. However, making the simplistic assumption that losses are deterministic gives better approximation of the real throughput since it introduces an error that cancels the error introduced by the assumption on the linearity of the rate increase. This cancellation does not appear on all connections. For example, it does not appear on our WAN connection where the problem of sub-linearity does not exist. On the WAN connection, it is better to model losses as they really are, this means as Poisson.

We conclude that the sub-linearity of the window increase in presence of linear models for TCP results in wrong conclusions on the way with which losses appear, and hence on the correctness of a given model for TCP congestion control. A model making wrong assumptions on losses may give better performance than a model making the correct assumptions. In the future, some works must be devoted to modeling the sub-linearity of TCP window evolution in local area networks otherwise the problem will always exist.

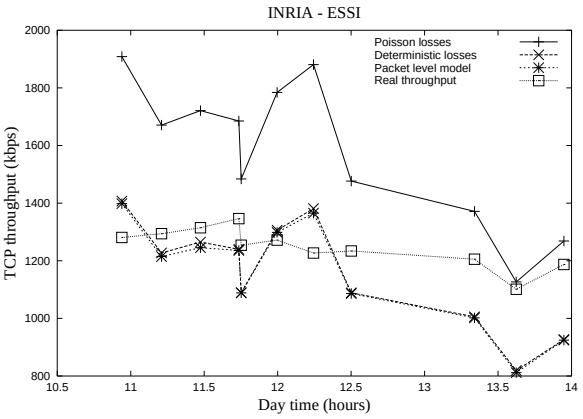


Figure 5.11: LAN: packet-level model

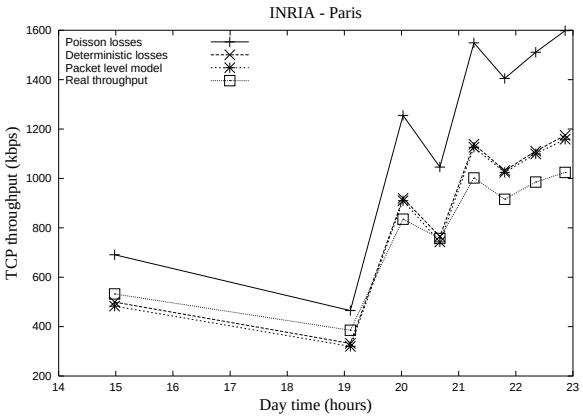


Figure 5.12: MAN: packet-level model

5.3.3 Comparison with packet-level approach

In addition to the consideration of all the functions of the loss process, our model contains two additional corrections: one for the timeout mechanism (3.1) and one for the packet nature of TCP (3.2). To validate these corrections, we compare the result of our model to that of the detailed packet-level model in [105] which is supposed to model correctly these two phenomena. We plot in Figures 5.11, 5.12 and 5.13 for respectively the LAN, MAN and WAN connections: the results of our model under deterministic and Poisson losses, the throughput from [105] and the real throughput. The throughput given by our linear fluid model is corrected by (3.1) and (3.2). The model in [105] considers deterministic losses (due to the mutual independence it assumes between the two processes $\{X_n^*\}$ and $\{S_n\}$) and usually it should give close performance to our model under deterministic losses. The three figures show that this is indeed the case. This proves that our model is a generalization of [105]. The generalization figures in the variance and correlation functions of the loss process we added to the expression of the throughput. The heuristics for the addition of timeouts and the packet nature of TCP are approximately the same in both models.

We notice in the figures that the model in [105] underestimates the throughput on the WAN connection where the loss process is Poisson rather than deterministic. Our model with Poisson losses gives better performance on this connection. Recall that the model in [105] as well as our model with deterministic losses estimate well the real throughput on the LAN connection due to the problem of error cancellation we already described.

5.3.4 Validation of bounds for the throughput

To validate the bounds for the throughput we found in the case of window limitation (Equations (5.25) and (5.26)), we consider the WAN connection where our linear model for TCP is appropriate. We plot in Figure 5.14 our results for the whole day. In the previous sections we

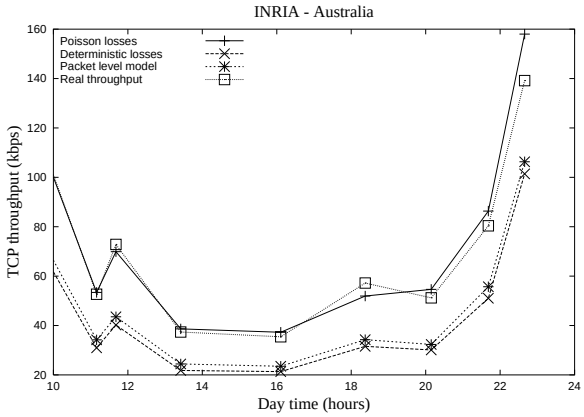


Figure 5.13: WAN: packet-level model

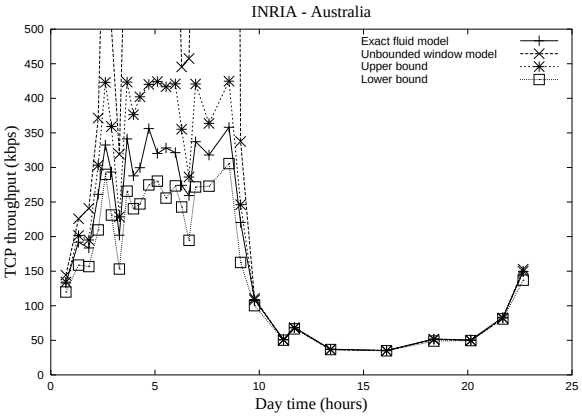


Figure 5.14: WAN: bounds for the throughput

only considered the working hours where the receiver window (equal to 32 Kbytes on the WAN connection) is not reached. The figure contains the throughput given by our model without window limitation (5.6), our two bounds and the throughput of the exact fluid model calculated numerically with a maximum limit of the transmission rate. First, we see that from 0 to 10 o'clock, the throughput calculated in the case of no limit on the congestion window significantly deviates from the exact fluid model. This deviation means that the receiver window is frequently reached during these hours of the day and our bounds can be then applied to estimate the throughput. Indeed, the figure shows that our bounds are quite close to the ideal throughput in this region. The figure also shows that from 10 to 0 o'clock all the lines coincide. During these latter hours, the receiver window is rarely reached and the result of our model in case of no limitation of the window (5.6) can be used to approximate the throughput of TCP. This result coincides in this region with our bounds for the throughput. Hence, our bounds form a good approximation of the throughout during the whole day.

5.4 Conclusions

In this chapter we presented an analysis of TCP throughput under a general loss process. The only assumption on the loss process under consideration is stationarity and ergodicity. We provided explicit expressions for the throughput in case of no limit on the transmission rate. The throughput is shown to be inversely proportional to the average round-trip time and to the square root of the packet loss probability, as was already observed for much simpler loss models [84, 92, 105]. The throughput is also a function of the variance and the correlation of inter-loss times. We further provided bounds for TCP throughput when a maximum limit exists on the congestion window.

The importance of our model is due to the different types of loss processes we observed when measuring Internet traffic. The model we proposed is able to capture any correlation or any

distribution of inter-loss times. Several existing models can be seen as particular cases of our general model. On paths where TCP transmission rate increases linearly between congestion moments, our model gives excellent results. However, on paths where TCP window growth is sub-linear, we noticed some overestimation of the real throughput. In a future work, we will try to account for this sub-linearity of TCP. The first problem with this sub-linearity is its modeling. It is not clear which additional variables have to be considered to quantify it. One possible direction could be the use of the correlation of window and round-trip time. The second problem will be of course the analysis of the sub-linear model for TCP. Another direction for future research is to provide algorithms for the estimation of the different functions of our model for losses (e.g., variance and correlation functions of inter-loss times) on runtime. One can use for this purpose averaging algorithms similar to those used by TCP for the estimation of the average round-trip time and its variance (see Section 2.1.3). One can also think about the application of our model to cases where we know *a priori* the reaction of the network to injected packets. For example, in a network where packets are dropped with a constant probability (e.g., random buffers, wireless networks), we can calculate the different functions of the loss process the connection will see, plug these functions in our general expression, and get the throughput of the TCP connection.

Chapter 6

Modeling TCP congestion control with window limitation

After the calculation of the throughput for a general loss process and an unlimited congestion window, we still have to investigate the case of window limitation. This will conclude the first part of this thesis which deals with end-to-end modeling of TCP congestion control. Recall that window limitation is the result of a small receiver window compared to the capacity of the network. Till now, we found approximations of the throughput in presence of such a limitation. The calculation of an explicit expression of TCP throughput for a general loss process seems to be impossible in this case given the sub-linearity of the resulting model. See Figures 3.4 and 6.1 for examples of window limitation. Clearly, the problem has a solution in the case of deterministic losses. The solution is given by Equation (5.20). It corresponds to the approximation of the throughput using the fixed-point approach. In [96], the authors model the case of Poisson losses but they have an error in the establishment of one of their equations which simplifies their analysis and gives a wrong expression for the throughput. It has been argued in [96] that the Poisson process is a good approximation of losses in the Internet. Indeed, we saw that it is the case in wide area networks (Section 3.3.1). To our knowledge, there is no other attempts to calculate the throughput of TCP in case of window limitation.

Now, the authors in [96] make a remarkable observation that the problem of TCP congestion control can be reformulated in terms of an equivalent M/G/1 queue [80], where the transmission rate of TCP is transformed into the workload of the queue and where the occurrence of losses corresponds to the arrival of clients. The service times of clients in the 'dual' queueing model are not i.i.d. (independent and identically distributed) but depend on the workload of the system. The calculation of the throughput of TCP is then transformed into the calculation of the time average of the workload of an M/G/1 queue. This reformulation of the problem has motivated us to use some techniques from queueing theory to calculate TCP throughput [16].

Although there is not much effort in the literature devoted to the calculation of the throughput of TCP in presence of window limitation, many works [81, 110] have studied queueing systems with service times that depend on the workload. Such systems are often called state-dependent

queues. The workload of a queueing system corresponds to the time required to serve all the clients present in the system [80]. In [81], an asymptotic approximation is used for solving state-dependent queues in which both inter-arrival times, service requirements and the service rate may depend on the workload. In [110], an implicit characterization of the steady state distribution is obtained. Closed-form expressions are obtained in this latter reference for special cases that do not cover our dual model for TCP congestion control. Later, we will explain the relation between the TCP congestion control problem and the queue workload problem. We just wish to recall here that an M/G/1 system means a queue with an infinite waiting room and a single server. Clients arrive to the queue according to a Poisson process and their service times can follow any distribution. In particular, service times can be dependent on each other which is the case of the dual model of TCP congestion control.

We address in this chapter the calculation of TCP throughput in presence of window limitation. For the analysis, we consider the same fluid model for TCP rate evolution: the transmission rate of the TCP connection $X(t)$ increases with a rate α between losses and decreases multiplicatively by a factor $\nu \in (0, 1)$ when a loss occurs. At any moment, the transmission rate of TCP cannot exceed a maximum value M. Once it reaches M, it stays at this value until the next loss event. We substitute the throughput obtained with this model into Equations (3.1) and (3.2) in order to account for the timeout mechanism and the packet nature of TCP.

Concerning the loss process, we consider a generalization of the Poisson process. We assume that loss events appear in batches with times between batches exponentially distributed. The number of losses within a batch follows a general distribution and is independent of the numbers of losses in other batches and on times between batches. This corresponds to the well known batch Poisson process [80]. Recall that by a loss we mean an event that causes the reduction of TCP window by a constant factor ν. It can be the result of one or more packets lost within the same round-trip time. More generally, a loss represents the moment at which the TCP source decides that the network is congested and that it has to reduce its window.

Our model for losses is first a generalization of the Poisson model in [96] for which the correct throughput of TCP is still not available. As claimed in [96] and as our measurements showed (Section 3.3.1), the Poisson process is a good approximation of losses in wide area networks. Second, this model permits to approximate the throughput on paths where the loss process is highly bursty as on our LAN connection (Figure 3.4). Indeed, if on our LAN connection we group together losses separated by less than 0.4s, we get approximately the batch Poisson process we already described (see [14] for details). Third, the batch Poisson model is useful for any future version of TCP that reduces its window as a function of the level of congestion of the network. For example, one can modify TCP in a way to reduce the window multiple times by a constant factor ν (less than one half) and this is as a function of the number of packets lost at the moment of congestion.

In the following, we calculate the throughput of TCP for this particular loss process. We

write the Kolmogorov equation of the rate in the stationary regime and we solve it using Laplace-Stieltjes Transforms. This gives us the expressions of all the moments of the transmission rates including the throughput. The analysis also gives us the distribution of the transmission rate in the stationary regime. We will see that even though the loss process is simple, the analysis and the expression of the throughput are quite complex which makes useful the approximations of the throughput we calculated in the previous chapters. The research work in this chapter has appeared in [16] and one can find more details on our analysis in [15].

6.1 Model and preliminary analysis

We already described our linear fluid model for the window evolution (Figure 5.2). We consider the transmission rate in our analysis and this is only for the purpose to make our results applicable to other flow control mechanisms that use a rate-based approach instead of a window-based approach (e.g., Available Bit Rate service in ATM networks [21]). At any moment, one can switch to the window space by simply multiplying the transmission rate of TCP by the average round-trip time RTT. For example, the time average of the transmission rate (the throughput) is equal to the time average of the window size divided by RTT. The distribution of the transmission rate at x is equal to the distribution of the window size at $xRTT$. Recall that the slope of the rate increase is equal to $\alpha = 1/(bRTT^2)$ and the multiplicative decrease factor upon a loss event is constant and equal to ν (typically, $\nu = 0.5$).

Let us now formulate our model for losses. We assume that a batch of loss events is the result of a single congestion event. Again, a loss for us is the moment at which the source reduces its window by a factor ν and one must not confuse it with the loss of packets. A congestion event results in a random number of losses (or a random number of window reductions by ν)[13] instead of a single loss event as in our previous models. For a batch of n losses, the window is reduced by a factor ν^n. Congestion events (or batches) are assumed to occur according to an independent Poisson process of intensity λ. We denote the sizes (i.e., the numbers of losses) of consecutive batches by $N_1, N_2, N_3, \ldots$, and we assume that these numbers constitute an i.i.d. sequence. The size of an arbitrary batch is generically denoted by $N \stackrel{d}{=} N_k$. The Poisson process and the sequence N_k, $k = 1, 2, \ldots$, are independent of each other and independent of the rate evolution.

Before passing to the analysis, let us introduce some further common notation. We denote the pgf (probability generating function) [80] of the distribution of N by

$$Q(z) = \mathrm{E}\left[z^N\right] = \sum_{n=1}^{\infty} q_n z^n, \quad |z| \leq 1. \tag{6.1}$$

$q_n, n \geq 1$, denotes the probability that a congestion event results in n losses (i.e., in n reductions

[13] The multiple reductions of the congestion window at the moment of congestion can be the result of packets lost in multiple consecutive round-trip times as we observed on our LAN connection (Figures 3.4). One can also think about a new TCP congestion control that reduces the window multiple times by a constant factor ν (less than one half) as a function of the level of congestion.

of the congestion window by ν). Note that our model with $\nu = 0.5$ and $q_1 = 1$ reduces to the model studied in [96], where congestion events appear according to a Poisson process and where the window is divided by two upon every congestion event.

Now, we recall the following stability result we established in Theorem 5.1. For any initial rate, there exists a stationary process $X^*(t)$ such that $X(t)$ converges to $X^*(t)$ in distribution. Moreover, we have almost sure,

$$\lim_{t\to\infty} \sup_{s\geq t} |X(s) - X^*(s)| = 0. \tag{6.2}$$

From now on, we put ourselves in the stationary regime which is unique and attained asymptotically for any initial rate according to (6.2). For $x \in (0, M]$, denote by $F(x)$ the distribution function of the process $X(t)$ in the stationary regime. It follows from Theorem 6.1 that this distribution is unique and is independent of $X(0)$. We first assume that $F(x)$ is continuous in $x \in (0, M)$. It is clear from physical considerations that $F(x)$ has an atom at $x = M$. This is because at any moment, there is a non-null probability that the transmission rate is equal to the maximum limit M. Under this assumption we find in the following a continuous function $F(x)$ which is an equilibrium distribution for the transmission rate and, hence, from its uniqueness it follows that it is the desired distribution. Instead of $F(x)$ it will be convenient to work with the complementary distribution function,

$$\overline{F}(x) = 1 - F(x) = \mathrm{P}\left\{X > x\right\}, \quad x \in (0, M].$$

6.2 Kolmogorov equation

The rate evolution $X(t)$ forms a Markov process since at any moment, we can predict the future of the rate from its current value without looking at the past [80]. With this in mind, we derive a steady-state Kolmogorov equation [80] for $\overline{F}(x) = \mathrm{P}\left\{X > x\right\}$ which will be the basis to our analysis. Recall that the Kolmogorov equation for a continuous time process corresponds to the relation between the distributions of the process at arbitrary two moments t and $t + \Delta$. We use the following up and down crossing argument. Assume that the process $X(t)$ is in equilibrium and consider a level $x \in (0, M)$. Whenever the rate increases from less than or equal to x to more than x we say that an up-crossing of the level x has occurred. Similarly, if the rate decreases from more than x to less than or equal to x we say that a down-crossing of the level x has occurred. Let $[t, t + \Delta]$ be a small time interval where t is a deterministic time moment. When the process is in equilibrium, the probability of up-crossing,

$$(1 - \lambda\Delta)\mathrm{P}\left\{x - \alpha\Delta < X \leq x\right\} + \mathrm{o}(\Delta),$$

is equal to the probability of down-crossing,

$$\lambda\Delta \sum_{n=1}^{\infty} q_n \mathrm{P}\left\{x < X \leq \min(\nu^{-n}x, M)\right\} + \mathrm{o}(\Delta).$$

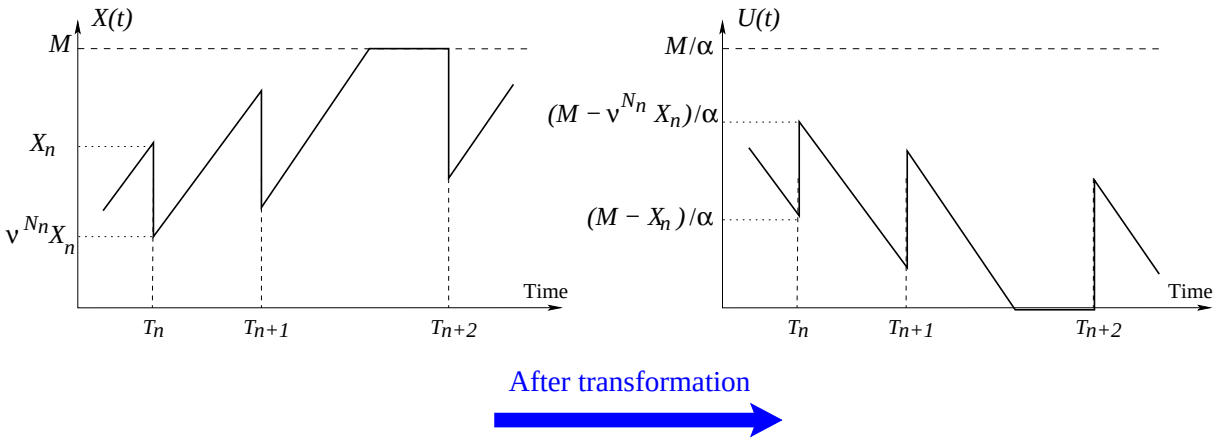


Figure 6.1: The dual queueing model

After equating these two probabilities, we pass $\Delta \downarrow 0$. Since we assumed that $F(x) = \mathrm{P}\{X \leq x\}$ is continuous for $x < M$ (see Remark 6.3 for a justification of this assumption), we conclude that the derivative of $F(x)$ exists and is continuous for all x except at $x = \nu^n M$ when $q_n > 0$. For $x \in (0, M) \backslash \{\nu^n M\}_{n=1,2,\ldots}$, we obtain the following steady-state Kolmogorov equation,

$$\alpha \frac{\mathrm{d}}{\mathrm{d}x} \mathrm{P}\{X \leq x\} = \lambda \sum_{n=1}^{\infty} q_n \mathrm{P}\left\{x < X \leq \min(\nu^{-n} x, M)\right\},$$

or equivalently,

$$-\alpha \frac{\mathrm{d}}{\mathrm{d}x} \overline{F}(x) = \lambda \left(\overline{F}(x) - \sum_{n=1}^{\infty} q_n \overline{F}(\min(\nu^{-n} x, M)) \right). \tag{6.3}$$

From this differential equation we shall determine $\overline{F}(x)$, $x \in (0, M)$, in terms of the probability

$$P_M = \mathrm{P}\{X = M\} = 1 - F(M^-) = \overline{F}(M^-).$$

In Section 6.4, we use (6.3) to determine the moments of the transmission rate in terms of P_M. Recall that the throughput of TCP coincides with the first moment of $X(t)$ (Equation (4.1)). Then, we find the distribution function itself in Section 6.5. The distribution of the window size at a value x will be equal to $F(xRTT)$. The unknown probability P_M is then determined using the fact that $\overline{F}(x)$ is a complementary probability distribution function ($\overline{F}(0) = 1$). However, the expression obtained for P_M in this way does not lend itself for computational purposes. Therefore, we show in Section 6.6 an elegant alternative to determine P_M which leads to an efficient and numerically stable algorithm for computations.

6.3 The dual M/G/1 queueing model

Before proceeding with determining the moments and the distribution of TCP rate, we briefly show how our problem can be related to an M/G/1 queueing problem with service times depend-

ing on the system workload, see also [96]. Define the following process,

$$U(t) = \frac{M - X(t)}{\alpha}. \tag{6.4}$$

I.e., $U(t)$ is obtained by 'flipping' $X(t)$ around a horizontal line (Figure 6.1). In particular, the area between $X(t)$ and the maximum rate M corresponds to the area below $U(t)$. Note that $U(t)$ resembles the evolution in time of the workload (or the virtual waiting time) in a queueing system [80]. A TCP rate equal to M corresponds to an empty queueing system. The linear increase in TCP rate between congestion events corresponds to the decrease in workload due to service in the M/G/1 model. The arrival of a batch of losses in the TCP model corresponds to the arrival of a batch of clients in the M/G/1 model. The reduction of the rate upon the arrival of a client corresponds to the increase in workload upon the arrival of a client. Given that the amount by which the rate is reduced upon the occurrence of a batch of losses depends on the current value of rate (and of course on the number of losses in the batch), the service time in the dual queueing model is dependent on the current workload. We conclude that the dual queueing model behaves indeed as an M/G/1 queue (infinite buffer capacity, one server and Poisson arrivals with intensity λ) with state-dependent service requirements. If U_n is the workload seen by the nth batch of clients in the M/G/1 queue, the service time x_n of the clients in the batch is equal to

$$x_n = \left(\frac{M}{\alpha} - U_n\right).\left(1 - \nu^{N_n}\right),$$

where N_n is the number of losses in the batch. Instead of working with the rate as we will do, one can solve the queueing problem (for moments and distributions) and switch back to the TCP problem by using Equation (6.4). In particular, $\mathrm{E}\left[X^k\right] = \mathrm{E}\left[(M - \alpha U)^k\right]$, $\mathrm{P}\left\{X \leq x\right\} = 1 - \mathrm{P}\left\{U \leq (M - x)/\alpha\right\}$ for $x < M$, and P_M is equal to the fraction of time the server in the dual queueing model is empty.

6.4 Moments of TCP rate

Using the Kolmogorov Equation (6.3), we calculate in this section the moments of the transmission rate of the TCP connection. The moments of the window size can be directly deduced by a multiplication by RTT. For example, the order k moment of $W(t)$ is equal to the order k moment of $X(t)$ multiplied by RTT^k. Of particular interest is the first moment of $X(t)$ which is equal to the throughput of the connection (Equation (4.1)). The second moment of $X(t)$ is also important since it tells us how much the rate of the connection varies. The different moments of the rate will be calculated as a function of P_M, the probability that the rate is equal to M at an arbitrary time moment, or in other words the fraction of time the rate stays at M.

Define for $\mathrm{Re}(s) \geq 0$ the LST (Laplace-Stieltjes Transform) of the rate distribution by

$$\hat{f}(s) = \int_{x=0}^{+\infty} \mathrm{e}^{-sx} \mathrm{d}F(x) = \int_{x=0}^{M+} \mathrm{e}^{-sx} \mathrm{d}F(x).$$

Taking LTs (Laplace Transforms) in (6.3) leads to

$$\alpha\left(\hat{f}(s) - P_M \mathrm{e}^{-sM}\right) = \lambda\frac{1-\hat{f}(s)}{s} - \lambda\sum_{n=1}^{\infty}\nu^n q_n \frac{1-\hat{f}(\nu^n s)}{\nu^n s}. \tag{6.5}$$

Note that (6.5) holds in particular for $M = \infty$, i.e., no limitation of TCP rate, in which case $P_M = 0$. Using $\mathrm{E}\left[X^k\right] \leq M^k$, $k = 1, 2, \ldots$, we may write

$$\hat{f}(s) = 1 + \sum_{k=1}^{\infty}\frac{(-s)^k}{k!}\mathrm{E}\left[X^k\right], \tag{6.6}$$

$$\frac{1-\hat{f}(\gamma^{-n}s)}{\gamma^{-n}s} = \sum_{k=0}^{\infty}\frac{(-\gamma^{-n}s)^k}{(k+1)!}\mathrm{E}\left[X^{k+1}\right]. \tag{6.7}$$

We use for these latter equations the Taylor series of $\hat{f}(s)$ and the fact that

$$\mathrm{E}\left[X^k\right] = (-1)^k \left.\frac{\mathrm{d}^k\hat{f}(s)}{\mathrm{d}s^k}\right|_{s=0},$$

$$\mathrm{E}\left[X^0\right] = 1.$$

Substituting (6.6) and (6.7) in (6.5), using the absolute convergence of the doubly-infinite series to interchange the order of summation, and equating the coefficients of equal powers of s we get, for $k = 1, 2, \ldots$,

$$\mathrm{E}\left[X^k\right] = \frac{k\alpha\left(\mathrm{E}\left[X^{k-1}\right] - P_M M^{k-1}\right)}{\lambda\left(1 - Q(\nu^k)\right)}, \tag{6.8}$$

from which the moments of the rate can be recursively obtained. In particular, we find for $k = 1, 2$,

$$\mathrm{E}\left[X\right] = \bar{X} = \frac{\alpha\left(1-P_M\right)}{\lambda\left(1-Q(\nu)\right)}, \tag{6.9}$$

$$\mathrm{E}\left[X^2\right] = \frac{2\alpha\left[\alpha\left(1-P_M\right) - \lambda P_M M\left(1-Q(\nu)\right)\right]}{\lambda^2\left(1-Q(\nu)\right)\left(1-Q(\nu^2)\right)}. \tag{6.10}$$

Later, we specify these results for the case $N \equiv 1$, this means for the case of a Poisson loss process.

The first two moments can also be obtained using direct arguments, see Remarks 6.1 and 6.2 below. Such arguments were also used by Misra et al. [96] for the case $\nu = 0.5$ and $N \equiv 1$. However in their analysis, an error appears which results in an additional equation besides (6.9) and (6.10) from which they determine an erroneous expression for the probability P_M, see Remark 6.2.

Remark 6.1 *The throughput (as a function of P_M) can be obtained by considering the mean drift. The upward drift of the rate is given by $\alpha\mathrm{P}\{X < M\}$ and the downward drift equals $\lambda\mathrm{E}[X]\left(1 - \mathrm{E}\left[\nu^N\right]\right)$. Equating these gives (6.9).*

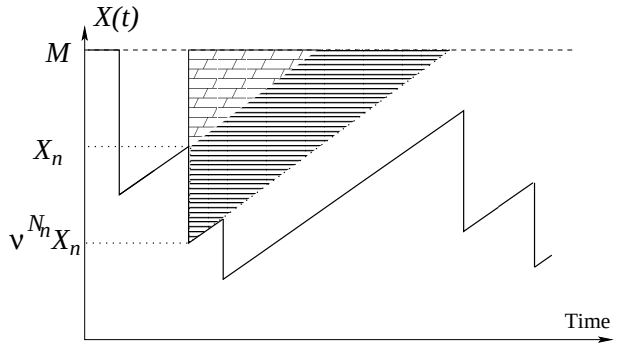


Figure 6.2: Area associated with a single congestion event

We can further derive $\mathrm{E}\left[X^2\right]$ *by applying an argument similar to Little's law [80] as was done by Misra et al. [96] for the case* $\nu = 0.5$ *and* $N \equiv 1$. *For details we refer to [15]. The main idea is sketched in the following. For the dual queueing model described in Section 6.3, we can equate the mean workload* $\mathrm{E}[U]$ *with* λ *times the mean area below* $U(t)$ *induced by a single batch arrival. We also use the PASTA (Poisson Arrivals See Time Averages) theory [80] to equate the moments of the workload upon arrivals with the moments of the workload at arbitrary time moments (*$\mathrm{E}[U_n] = \mathrm{E}[U(t)]$*). Back to the original model, the mean surface of the area* above $X(t)$ *in Figure 6.2 equals* $M - \mathrm{E}[X]$. *The expected surface of the area induced by a single congestion event (the surface of the larger triangle minus that of the smaller one) is equal to*

$$\frac{1}{2\alpha}\left[\left(Q(\nu^2)-1\right)\mathrm{E}\left[X^2\right] - 2M\left(Q(\nu)-1\right)\mathrm{E}[X]\right].$$

Multiplying this surface by λ, *equating the result with* $M - \mathrm{E}[X]$, *and using (6.9) indeed gives (6.10).*

Remark 6.2 *For a special case of our model (*$\nu = 0.5$, $N \equiv 1$*), yet another way is pursuit in [96] to derive (6.9) and (6.10). However, there, the final result is incorrect due to a small error in an intermediate step. First, let us put ourselves in the transitory regime. Defining* $P_M(t) = \mathrm{P}\{X(t) = M\}$, $\mathrm{E}[X(t)]$ *and* $\mathrm{E}\left[X(t)^2\right]$ *satisfy*

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{E}[X(t)] = -\lambda\left(1-Q(\nu)\right)\mathrm{E}[X(t)] + \alpha\left(1-P_M(t)\right),$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{E}\left[X(t)^2\right] = -\lambda\left(1-Q(\nu^2)\right)\mathrm{E}\left[X(t)^2\right] + 2\alpha\left(\mathrm{E}[X(t)] - MP_M(t)\right).$$

We refer to [96] for more details on the establishment of these differential equations. In steady state we have $\mathrm{E}[X(t)] \equiv \mathrm{E}[X]$, $\mathrm{E}\left[X(t)^2\right] \equiv \mathrm{E}\left[X^2\right]$ *and* $P_M(t) \equiv P_M$. *The terms on the left-hand side of the above two equations converge to zero. Writing the right-hand terms in the stationary regime gives (6.9) together with*

$$0 = -\lambda\left(1-Q(\nu^2)\right)\mathrm{E}\left[X^2\right] + 2\alpha\left(\mathrm{E}[X] - P_M M\right). \tag{6.11}$$

When substituting $\mathrm{E}[X]$ *by its expression in (6.9), this leads to (6.10). For the case* $\nu = 0.5$ *and* $N \equiv 1$*, the formula given in [96] for* $\mathrm{E}\left[X^2\right]$ *(below Equation (4) in that reference) differs from (6.11) by a factor* $-\alpha = -1/RTT$*. This resulted in a third (incorrect) equation which is linearly independent of (6.9) and (6.10) from which* P_M *was determined simultaneously (but erroneously) with* $\mathrm{E}[X]$ *and* $\mathrm{E}\left[X^2\right]$*. In Section 6.6, we will show how* P_M *can be determined correctly and computed efficiently. The moments of the rate can be then calculated using recurrent Equation (6.8).*

6.5 Distribution function of TCP rate

We profit from the Kolmogorov equation we established in (6.3) to calculate the distribution function of TCP rate, and hence of TCP window. Even though there is no direct application of the distribution for the moment, we do the calculation for the completeness of the study. One possible application of the distribution could be the calculation of P_M. However, we will see that this is not an efficient method, see Remark 6.4. In the next section, we provide a more efficient method for the calculation of P_M. We start first by the case of finite M. We provide the expression of the distribution in every interval $[\nu^k M, \nu^{k-1} M]$ with $k = 1, 2, \ldots$. Then, for the case $M = \infty$, we give an expression of the distribution for any $x > 0$ as an infinite sum of exponentials.

6.5.1 Rate distribution for finite M

For $\nu M \leq x < M$, Equation (6.3) reduces to

$$-\alpha \frac{\mathrm{d}}{\mathrm{d}x} \overline{F}(x) = \lambda \overline{F}(x).$$

Hence,

$$\overline{F}(x) = P_M \mathrm{e}^{\frac{\lambda}{\alpha}(M-x)}, \quad \nu M \leq x < M. \tag{6.12}$$

To find the entire distribution, we introduce for $k = 1, 2, 3, \ldots$, the set of functions

$$\overline{F}_k(x) = \overline{F}(x), \quad \nu^k M \leq x \leq \nu^{k-1} M. \tag{6.13}$$

A function $\overline{F}_k(x)$ is equal to $\overline{F}(x)$ within the interval $[\nu^k M, \nu^{k-1} M]$. Outside this interval, it can take any value. In particular, it can be equal to $\overline{F}(x)$. Equation (6.3) can now be written as

$$\frac{\mathrm{d}}{\mathrm{d}x} \overline{F}_k(x) = -\frac{\lambda}{\alpha} \overline{F}_k(x) + \frac{\lambda}{\alpha} \sum_{n=1}^{k-1} q_n \overline{F}_{k-n}(x/\nu^n). \tag{6.14}$$

Since $\overline{F}(x)$ is continuous (assumed to be continuous until Remark 6.3) for $0 < x < M$, we have

$$\overline{F}_k(\nu^{k-1} M) = \overline{F}_{k-1}(\nu^{k-1} M), \qquad k = 2, 3, \ldots. \tag{6.15}$$

$\overline{F}_k$ is recursively given by

$$\overline{F}_k(x) = \overline{F}_{k-1}(\nu^{k-1}M)\mathrm{e}^{\frac{\lambda}{\alpha}(\nu^{k-1}M-x)} - \frac{\lambda}{\alpha}\mathrm{e}^{-\frac{\lambda}{\alpha}x}\int_{u=x}^{\nu^{k-1}M}\mathrm{e}^{\frac{\lambda}{\alpha}u}\sum_{n=1}^{k-1}q_n\overline{F}_{k-n}(u/\nu^n)\mathrm{d}u. \tag{6.16}$$

We use for (6.16) the fact that the solution of the differential equation $f'(x)+af(x)=b(x)$ can be written in the following form

$$f(x) = f(x_0)\mathrm{e}^{-a(x-x_0)} - \mathrm{e}^{-ax}\int_x^{x_0}\mathrm{e}^{au}b(u)du.$$

We conclude from recursion (6.16) that a solution to (6.14) and (6.15) has the following form

$$\overline{F}_k(x) = P_M\sum_{i=1}^{k}c_i^{(k)}\mathrm{e}^{-\frac{\lambda x}{\alpha\nu^{i-1}}}, \quad k=1,2,... \tag{6.17}$$

This can be simply proved by induction on k. To determine the coefficients $c_i^{(k)}$, $1\le i\le k$, we first substitute (6.17) in (6.14). Then, equating terms with the same exponent, we get the following recursive formula

$$c_{i+1}^{(k)} = \frac{\nu^i}{\nu^i-1}\sum_{n=1}^{i}q_n c_{i-n+1}^{(k-n)}, \quad i=1,...,k-1. \tag{6.18}$$

We conclude that the coefficients of $\overline{F}_k(x)$ are a function of the coefficients of the functions with lower indices. We start from the function with $k=1$ in the calculation of coefficients. For a certain k, the recurrent Equation (6.18) gives us all the coefficients of $\overline{F}_k(x)$ starting from $i=2$. Given the coefficients $c_i^{(k)}, i=2,...,k$, the remaining coefficient $c_1^{(k)}$ can be determined from (6.15) and (6.17),

$$c_1^{(k)} = e^{\frac{\lambda}{\alpha}\nu^{k-1}M}\left[\sum_{i=1}^{k-1}c_i^{(k-1)}e^{-\frac{\lambda}{\alpha}\nu^{k-i}M} - \sum_{i=2}^{k}c_i^{(k)}e^{-\frac{\lambda}{\alpha}\nu^{k-i}M}\right]. \tag{6.19}$$

Once all the coefficients of $\overline{F}_k(x)$ are calculated, we can pass to the calculation of the coefficients of $\overline{F}_{k+1}(x)$ and so on. Note that to compute the different coefficients of the distribution function, we *do not* need P_M. Hence, one can use these coefficients for the calculation of P_M using that $\overline{F}(x)$ is a complementary distribution function. P_M can be determined by

$$\lim_{k\to\infty}\overline{F}_k(\nu^{k-1}M) = P_M\left(\lim_{k\to\infty}\sum_{i=1}^{k}c_i^{(k)}\mathrm{e}^{-\frac{\lambda}{\alpha}\nu^{k-i}M}\right) = 1. \tag{6.20}$$

However, this relation is not suitable to compute P_M, see Remark 6.4 below.

Remark 6.3 *With (6.13) and (6.17) we have found an equilibrium distribution function $F(x)$ satisfying (6.3) and continuous on the interval $(0,M)$. From Theorem 6.1, there is only one distribution function in the stationary regime and, hence, the assumption that $F(x)$ is continuous for $x<M$ is justified.*

Remark 6.4 *Recursion (6.18) is suitable to determine the distribution function on an interval $\nu^k M \leq x \leq M$ when k is not too large. For large k the recursion may become instable, since it involves subtraction of numbers of the same order. Therefore (6.20) is not suitable to compute P_M. In Section 6.6 below we will derive an alternative expression for P_M, which leads to a numerically stable and efficient algorithm to compute P_M.*

6.5.2 Rate distribution for infinite M

In this case, the results derived in the previous section cannot be applied immediately by letting M go to infinity. However, one can use the expression of the Laplace Transform for the calculation of the cumulative distribution. Using (6.5), an explicit expression of the Laplace Transform $F(x)$ can be found in this case which by inversion gives us the distribution of the rate as an infinite sum of exponentials. Indeed, when $M = \infty$, (6.5) becomes

$$\hat{f}(s) = -\frac{\lambda}{\alpha}\left[\frac{\hat{f}(s)}{s} - \sum_{n=1}^{\infty} q_n \frac{\hat{f}(\nu^n s)}{s}\right],$$

or equivalently,

$$\hat{f}(s) = \frac{\frac{\lambda}{\alpha}}{s + \frac{\lambda}{\alpha}} \sum_{n=1}^{\infty} q_n \hat{f}(\nu^n s). \tag{6.21}$$

Substituting the above equation repeatedly into itself l times, applying partial fraction expansion at each step [80], and then taking $l \to \infty$, we conclude that $\hat{f}(s)$ can be expressed as follows

$$\hat{f}(s) = \sum_{i=0}^{\infty} c_i \frac{-\frac{\lambda}{\alpha\nu^i}}{s + \frac{\lambda}{\alpha\nu^i}}, \tag{6.22}$$

for certain coefficients c_i (this is formally justified later). To determine the constants $c_i, i = 0, 1, ...,$ we substitute (6.22) into (6.21) and we equate the coefficients multiplying the terms $1/(s + \frac{\lambda}{\alpha\nu^i})$. This leads to the following recursive formula

$$\frac{c_i}{c_0} = \frac{\nu^i}{\nu^i - 1} \sum_{k=1}^{i} q_k \frac{c_{i-k}}{c_0}, \tag{6.23}$$

which determines the ratios c_i/c_0 for all $i = 0, 1, \ldots$. It is for this reason that both sides of the formula contain a factor $1/c_0$. The coefficient c_0 follows from $\hat{f}(0) = \overline{F}(0) = -\sum_{i=0}^{\infty} c_i = 1$,

$$c_0 = -(1 + \sum_{i=1}^{\infty} \frac{c_i}{c_0})^{-1}. \tag{6.24}$$

It can be easily shown that the infinite sum in the expression of c_0 converges. One can look at the Appendix at the end of this chapter for a proof of the convergence. Inverting back (6.22) into the time domain gives

$$F(x) = C + \sum_{i=0}^{\infty} c_i \mathrm{e}^{-\frac{\lambda}{\alpha\nu^i}x}, \tag{6.25}$$

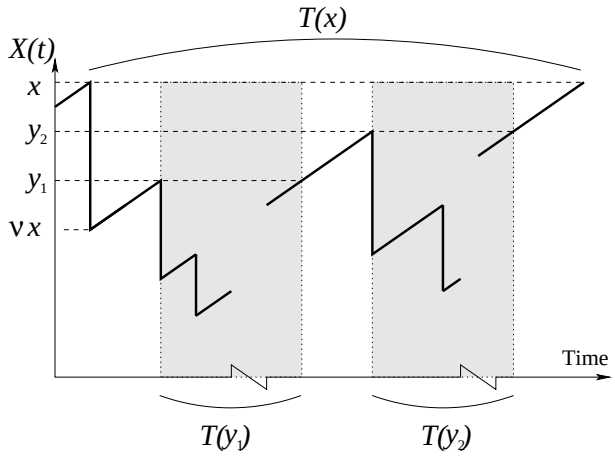


Figure 6.3: Random variable $T(x)$

with $C = 1$ so that $F(0) = 0$. Note that the infinite series in (6.25) is absolutely convergent for any value of $x \in [0, \infty)$. Thus, it is the unique solution to (6.3) when $M = \infty$. Recall that according to Theorem 6.1, the differential Equation (6.3) has a unique solution for any value of M. For the case of no limitation on the rate and $N \equiv 1$, the expression of $F(x)$ in the way we presented in (6.25) was already obtained in [104].

6.6 The probability of being at maximum rate

In Sections 6.4 and 6.5 we determined the moments and the distribution of TCP rate as a function of P_M, the probability that the rate is equal to M at an arbitrary time moment in the stationary regime. In this section we derive an expression for P_M from which it can be computed efficiently. To this end, we introduce the random variable $T(x)$ which is the time until the rate returns to the value x, starting just after a congestion event occurs with the rate being equal to $x \in (0, M]$ (Figure 6.3). We denote the expectation of $T(x)$ by $E(x) = \mathrm{E}[T(x)]$, $x \in (0, M]$. Then, from elementary renewal theory [80],

$$P_M = \mathrm{P}\{X = M\} = \frac{1/\lambda}{1/\lambda + E(M)}. \tag{6.26}$$

We proceed now to the calculation of $E(x)$. Consider Figure 6.3 for the illustration of the analysis. In this figure, we show a typical evolution of TCP rate starting from a congestion event that appears with the rate being equal to x until the moment the rate returns to x. Other congestion events may appear between these two moments. For simplicity, the figure only depicts congestion events having $N = 1$. The times to recover from intermediate congestion events are partly cut out of the picture (denoted by the shaded areas). Let us focus on the analysis of the general case when a congestion event results in multiple losses. Suppose for the moment that the initial congestion event (at rate x) was such that $N = n$. Let $T_n(x)$ be the time to get back

to rate x conditional on $N = n$ and we further write $E_n(x) = \mathrm{E}[T_n(x)] = \mathrm{E}[T(x)|N = n]$. Note that,

$$E(x) = \sum_{n=1}^{\infty} q_n E_n(x). \tag{6.27}$$

If no congestion events occur during the time $T_n(x)$ then $T_n(x) = (1-\nu^n)x/\alpha$, i.e., the rate x is reached in a straight line from the starting point at $\nu^n x$ (νx in the figure). Each time a loss occurs at a level $y \in (\nu^n x, x)$, it takes $T(y)$ time units to get back to rate y. Because of the memoryless property of the Poisson process, if we take out the shaded areas in Figure 6.3 and concatenate the non-shaded areas then the cut points (where the shaded areas used to be) form a Poisson process on the straight line from $\nu^n x$ to x. Thus if the cut points are given by $y_1, y_2, \ldots, y_m$ (in the figure we show for simplicity the case of two cut points) then

$$E_n(x) = \frac{(1-\nu^n)x}{\alpha} + E(y_1) + E(y_2) + \cdots + E(y_m).$$

Since the process of congestion events is a Poisson process, the mean number [14] of cut points is equal to $\lambda(1-\nu^n)x/\alpha$, and the position of each of the points y_j is uniformly distributed over the interval $(\nu^n x, x)$, see for instance [124, Theorem 1.2.5] [15]. Hence,

$$\begin{aligned} E_n(x) &= \frac{(1-\nu^n)x}{\alpha} + \mathrm{E}[m] \int_{y=\nu^n x}^{x} \frac{E(y)}{(1-\nu^n)x} \mathrm{d}y \\ &= \frac{(1-\nu^n)x}{\alpha} + \lambda \frac{(1-\nu^n)x}{\alpha} \int_{y=\nu^n x}^{x} \frac{E(y)}{(1-\nu^n)x} \mathrm{d}y \\ &= \frac{(1-\nu^n)x}{\alpha} + \frac{\lambda}{\alpha} \int_{y=\nu^n x}^{x} E(y) \mathrm{d}y. \end{aligned}$$

Using (6.1) and (6.27), we now arrive at

$$E(x) = \frac{(1-Q(\nu))x}{\alpha} + \frac{\lambda}{\alpha} \sum_{n=1}^{\infty} q_n \int_{y=\nu^n x}^{x} E(y) \mathrm{d}y. \tag{6.28}$$

This implicit equation for $E(x)$ is valid for all $0 < x \leq M$. It is also valid for all $x > 0$ in the case of no limitation of TCP rate. We can prove that the solution to (6.28) is unique, see [15] for details. Let us find the expression of $E(x)$. To this end, we define the Laplace Transform

$$\hat{e}(s) = \int_{x=0}^{\infty} \mathrm{e}^{-sx} E(x) \mathrm{d}x.$$

In [15], we showed that $\hat{e}(s) < \infty$ for $\mathrm{Re}(s) > \lambda/\alpha$. First, we apply Laplace Transform to (6.28). Using that the q_n and $E(x)$ are non-negative, we may interchange the order of integration and summation (twice), finally arriving at

$$\hat{e}(s) = \frac{1}{\alpha s - \lambda} \left(\frac{1-Q(\nu)}{s} - \lambda \sum_{n=1}^{\infty} q_n \hat{e}(s/\nu^n) \right). \tag{6.29}$$

[14] For a Poisson point process of intensity λ, the mean number of points that appear in a time interval T is equal to λT.

[15] Given that a point of a Poisson point process appears in an interval [0,T], the probability that this point appears between 0 and x with $x \in [0, T]$ is simply equal to x/T.

Repeated substitution of this equation into itself and applying partial fraction expansion [80] leads us to the following candidate solution,

$$\hat{e}(s) = \frac{1-Q(\nu)}{s} \sum_{i=0}^{\infty} \frac{e_i}{\nu^{-i}\alpha s - \lambda}, \tag{6.30}$$

where the e_i are constants to be determined. This representation will be justified by showing that it leads us to the unique solutions to (6.28) and (6.29). Substituting (6.30) into (6.29) and equating the coefficients multiplying the terms $1/(\nu^{-i}\alpha s - \lambda)$ leads to

$$\frac{e_i}{e_0} = \frac{1}{1-\nu^i} \sum_{n=1}^{i} \nu^n q_n \frac{e_{i-n}}{e_0}, \qquad i \geq 1, \tag{6.31}$$

$$e_0 = \left(1 + \sum_{n=1}^{\infty} \nu^n q_n \sum_{j=0}^{\infty} \frac{e_j/e_0}{\nu^{-j-n}-1}\right)^{-1}. \tag{6.32}$$

We note that the ratios e_i/e_0 are non negative and can be computed recursively from (6.31). The normalizing constant $e_0 > 0$ can be computed from (6.32). From (6.31), it can be shown (by induction on i) that

$$e_i \leq \nu^i e_0, \quad i = 1, 2, \ldots. \tag{6.33}$$

I.e., the e_i decay exponentially fast in i as $i \to \infty$. Hence, the infinite series in (6.32) converges fast and all the e_i exist. Moreover, the right-hand side of (6.30) certainly converges for $s > \lambda/\alpha$ and, from its construction, (6.30) is the unique solution to (6.29). By partial fraction expansion, (6.30) can be rewritten as

$$\hat{e}(s) = \frac{1-Q(\nu)}{\lambda} \sum_{i=0}^{\infty} e_i \left(\frac{1}{s - \nu^i \frac{\lambda}{\alpha}} - \frac{1}{s}\right).$$

Inverting this Laplace Transform back into the rate domain gives

$$E(x) = \frac{1-Q(\nu)}{\lambda} \sum_{i=0}^{\infty} e_i \left(\mathrm{e}^{\nu^i \frac{\lambda}{\alpha} x} - 1\right). \tag{6.34}$$

Using this expression of $E(x)$ in (6.26), we get

$$P_M = \left(1 + (1-Q(\nu)) \sum_{i=0}^{\infty} e_i \left(\mathrm{e}^{\nu^i \frac{\lambda}{\alpha} M} - 1\right)\right)^{-1}. \tag{6.35}$$

With the expression of P_M, we conclude the calculation of the moments and the distribution of TCP rate. Note that because of (6.33) and

$$\left(\mathrm{e}^{\nu^i \frac{\lambda}{\alpha} M} - 1\right) \sim \nu^i \frac{\lambda}{\alpha} M, \quad i \to \infty,$$

P_M can be computed efficiently from (6.35).

6.7 Particular case of a Poisson loss process

We specify in this section our results to the particular case of a Poisson loss process: congestion events occur according to a Poisson process and a congestion event results in only one loss, or in other words in only one reduction of the rate by ν. For $\nu = 0.5$, this becomes similar to the model in [96]. The results for this particular case can be obtained by taking $N \equiv 1$ ($q_1 = 1$, and $q_n = 0$ for $n = 2, 3, \ldots$) in our batch model. The pgf of the distribution of the batch size is equal here to $Q(z) = z$. Using this particular case, we compare in Section 6.8.2 our model to measurements from the Internet. In particular, we validate the throughput of TCP given by our model as well as the distribution of the transmission rate. We will consider for this purpose long distance TCP connections where the loss process is known to be close to Poisson.

From (6.9) and (6.10), we obtain the expressions of the first two moments of the rate of the TCP connection. Recall that the first moment of the rate coincides with the throughput.

$$\mathrm{E}\left[X\right] = \bar{X} = \frac{\alpha}{(1-\nu)\lambda}(1 - P_M).$$

$$\mathrm{E}\left[X^2\right] = \frac{2\alpha[\alpha(1-P_M) - \lambda P_M M(1-\nu)]}{\lambda^2(1-\nu)(1-\nu^2)}.$$

Note that for $P_M = 0$ (infinite receiver window), we get the same throughput as that given by our general model in the previous chapter (Equation (5.13)).

For finite M, the distribution function $F(x)$ can be successively computed on the intervals $[\nu^k M, \nu^{k-1} M], k = 1, 2, \ldots$, using (6.17). Recursion (6.18), which gives the coefficients $c_i^{(k)}$, reduces to

$$c_{i+1}^{(k)} = \frac{\nu^i}{\nu^i - 1} c_i^{(k-1)}, \quad i = 1, ..., k-1,$$

and $c_1^{(k)}$ is given by (6.19). When $M = \infty$ and for all $x > 0$, the distribution function is given by the sum of exponentials in (6.25), where

$$c_i = \frac{\nu^i}{\nu^i - 1} c_{i-1}, \qquad i = 1, 2, \ldots,$$

and c_0 is given by (6.24). Finally,

$$P_M = \left((1-\nu) \sum_{i=0}^{\infty} e_i \mathrm{e}^{\nu^i \frac{\lambda}{\alpha} M} \right)^{-1}, \tag{6.36}$$

where the coefficients e_i are given by

$$\begin{aligned} \frac{e_i}{e_0} &= \frac{\nu}{1-\nu^i} \frac{e_{i-1}}{e_0}, \quad i = 1, 2, \ldots, \\ e_0 &= \left(1 + \sum_{i=1}^{\infty} \nu^i \frac{e_i}{e_0} \right)^{-1}. \end{aligned}$$

For this particular form of P_M, we used the following equation that we established in an Appendix in [15] for the case of a Poisson loss process,

$$\sum_{i=0}^{\infty} e_i = \frac{1}{1-\nu}.$$

6.8 Model validation

Now, we pass to the validation of the results of our model. We consider the particular case of Poisson losses ($N \equiv 1$). A Poisson process approximates well the moments of losses on long distance TCP connections. This can be explained by the multiplexing of traffic and the superposition of the loss processes seen by the connection in the routers it crosses. The factor ν is set to 0.5. First, we consider our WAN connection (INRIA - University of South Australia) for the validation of our calculation of the throughput. We already saw that the loss process on this connection is close to Poisson (Section 3.3.1). During the night, we noticed that the congestion window reaches frequently the receiver window which justifies the use of our present model. The receiver window on the WAN connection is equal to 32 Kbytes (22 packets).

Second, we validate our calculation of the distribution of the window size. We run a particular connection for this validation. It is another WAN connection between INRIA in the south of France and a machine at the University of Michigan in the US. Our objective from running a particular connection for the validation of the distribution is that we want to show how the distribution looks for different receiver window sizes. Changing the receiver window on the connection to the University of South Australia was not possible for practical reasons. We run three times the second WAN connection for three receiver windows of 32, 48 and 64 Kbytes (resp. 22, 33 and 44 packets). Every run is of approximately one hour. We changed the receiver window by changing the buffer size of the socket interface at the receiver. This gave us three trace files. Every trace file corresponds to a particular receiver window. A trace file is generated by a tool we developed and we run at INRIA and contains a set of statistics on the connection as the number and moments of loss events, the evolution of the window size, the number of packets transmitted, etc. For every trace file, we plot the theoretical distribution in the case of a Poisson loss process (Section 6.7) and the one obtained from measurements. In our plots, we will show the distribution of the window size which we recall can be deduced from the distribution of the rate by a simple transformation using the average round-trip time.

Before presenting the results from measurements, we shall plot some numerical results to show how the distribution of the window looks (theoretically) in the stationary regime.

6.8.1 Numerical results

Consider the case of a long-life TCP connection with packets of size 1460 bytes and a constant round-trip time of one second. Using the results of Section 6.7, we compute the cumulative distri-

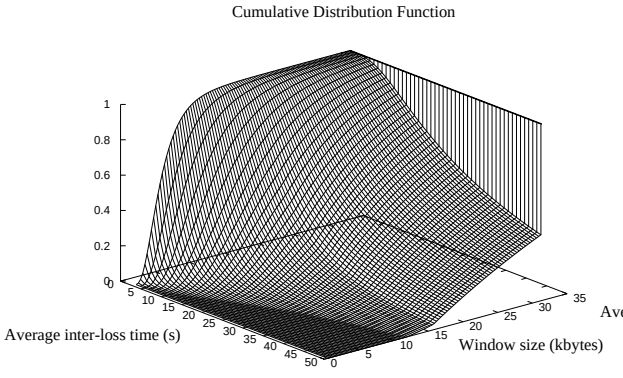


Figure 6.4: Distribution: $M = 32$ Kbytes

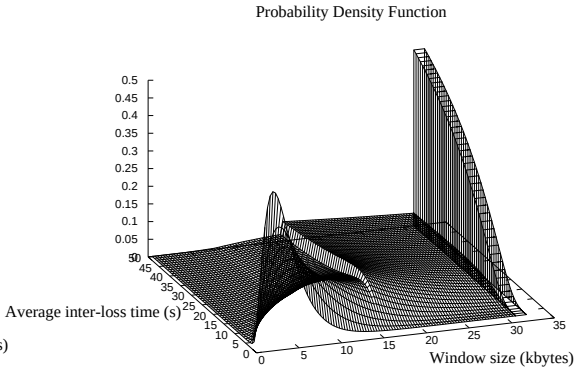


Figure 6.5: Density: $M = 32$ Kbytes

bution function $F(x)$ of the window size and its probability density function $f(x)$ for increasing values of the intensity of losses λ. We consider the two cases: $M = 32$ Kbytes and $M = \infty$. In Figures 6.5 and 6.7 we plot our results for the density function $f(x)$. The plots for the distribution function are shown in Figures 6.4 and 6.6. For $M = 32$ Kbytes, we calculate first P_M, the probability that the receiver window is reached (6.36). This calculation involves two infinite series ((6.31) and (6.32)), but as we said before, these series converge quite fast. Once P_M is computed, we calculate the distribution function successively on the intervals $[M/2, M], [M/4, M/2]$ and so on. For an infinite receiver window, we calculate first the coefficients c_i (6.23), then we find the distribution function for all $x > 0$ using the infinite sum of exponentials in (6.25). Again, this infinite sum converges quite fast.

First, we notice that the distribution function is continuous for all $x \in (0, M)$. It presents a jump at $x = M$ equal to P_M. For an infinite M, the distribution function is continuous for all $x > 0$. Now, when M is finite, the density function is discontinuous at $x = M/2$ especially for large inter-loss times. It presents a pulse at $x = M$ and this pulse is depicted in the figure by an area equal to P_M. When $M = \infty$, the density function exhibits neither pulses nor discontinuities.

6.8.2 Experimental results

First, we solve our present model for the throughput of TCP and we compare the result to what we get on our connection to Australia. The loss process is assumed to be Poisson. The result of the model is compared to the throughput of ideal TCP or the so-called exact fluid model. Figure 6.8 shows the results for the whole day. The figure also shows the two bounds for the throughput we calculated in Chapter 5. Every point in the figure corresponds to a trace file of approximately one hour. We see clearly how the result of our present model matches the throughput of ideal TCP and how it fells between our previous two bounds for the throughput.

Next, we validate our calculation of the distribution of TCP window using the traces of our

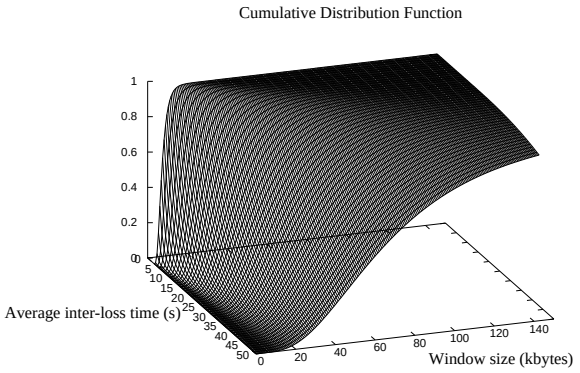


Figure 6.6: Distribution: $M = \infty$

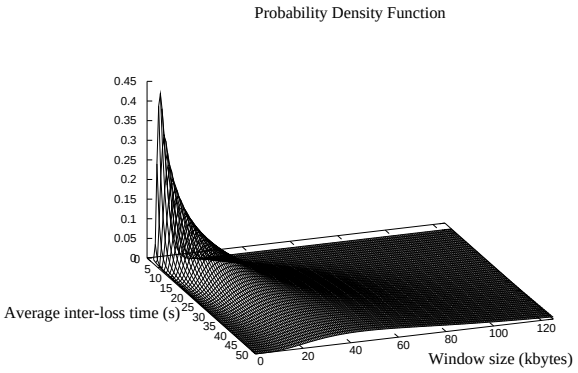


Figure 6.7: Density: $M = \infty$

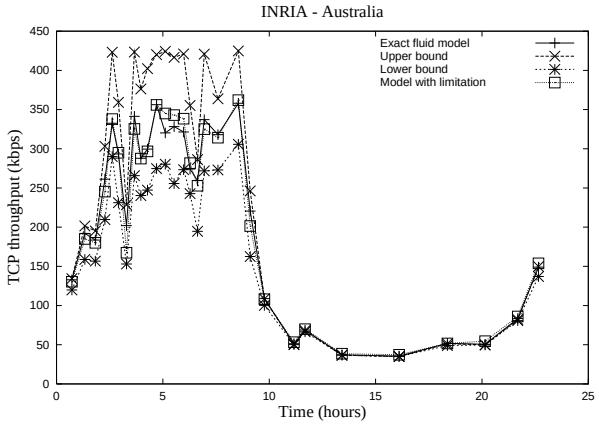


Figure 6.8: Validation of the throughput

second connection to Michigan State University. For the three values of M we cited above, we plot the real and theoretical distributions of TCP window. The results are presented in Figures 6.9, 6.10 and 6.11. When M is small, we observe a good match between the measured distribution and the one resulting from our model. For larger values of M, the difference between the two increases. In particular, as M increases, the measured probability density concentrates around the average window size. This deviation can be explained from the measured inter-loss time distribution. In Figure 6.13, we plot this distribution for M =32 Kbytes. This distribution is in agreement with an exponential law, resulting in the good match we observed between the model and the measurements. Figures 6.13 and 6.14 show the measured inter-loss time distribution for the other two values of M. We notice that the loss process is no longer Poisson, but closer to a deterministic process. This results in the degradation of the correspondence between our model and the measurements.

One explanation of the deviation of the loss process from a Poisson process for larger values

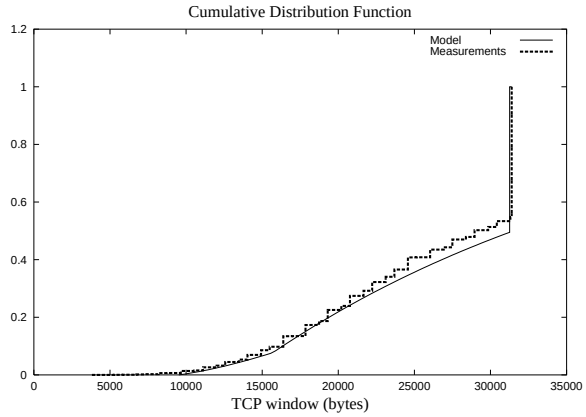


Figure 6.9: Distribution: $M = 32$ Kbytes

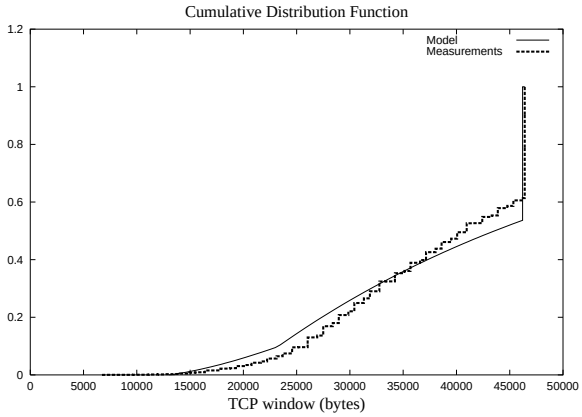


Figure 6.10: Distribution: $M = 48$ Kbytes

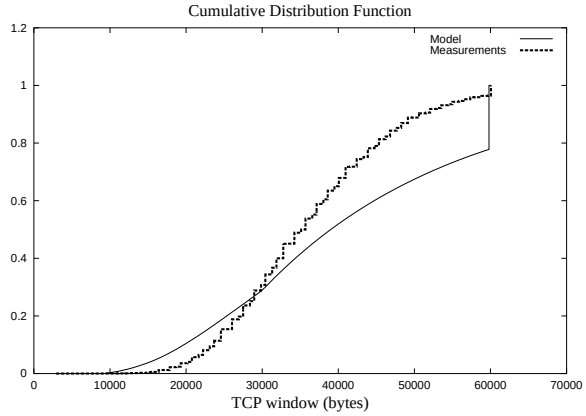


Figure 6.11: Distribution: $M = 64$ Kbytes

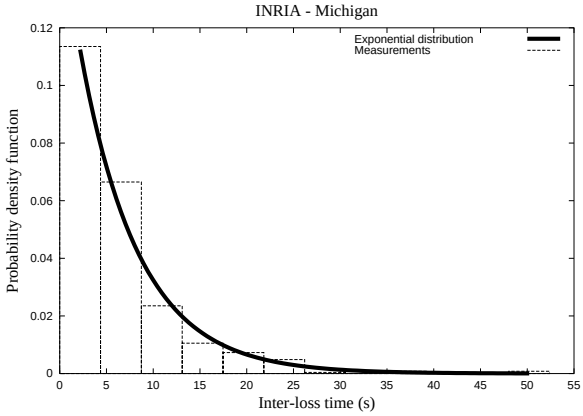


Figure 6.12: Inter-loss times: $M = 32$ Kbytes

of M is the following. A true Poisson loss process implies that the time until the next loss event is independent of the past. This is the case when the congestion of the network is dominated by the exogenous traffic and not dependent on our connection. This corresponds to the case when our connection's share of the available bandwidth on the path is small compared to that of the exogenous traffic, and when the packets of our connection are dropped in multiple congested routers along the path. A small M limits the bandwidth share of our connection and limits its impact on the network, resulting in a loss process close to Poisson. However for large M, our connection realizes a larger share of the bandwidth and thus contributes more to the congestion of network routers. When it reduces its window, the state of the network changes and becomes under-loaded. Some time is needed for the network to be loaded again, and during this time, the probability to get a loss event is very small. This is the reason for which small inter-loss times start to get small probabilities as M increases. In such a case when the loss process is close to a deterministic process, a simple approximation using the fixed-point approach [11, 105] could be

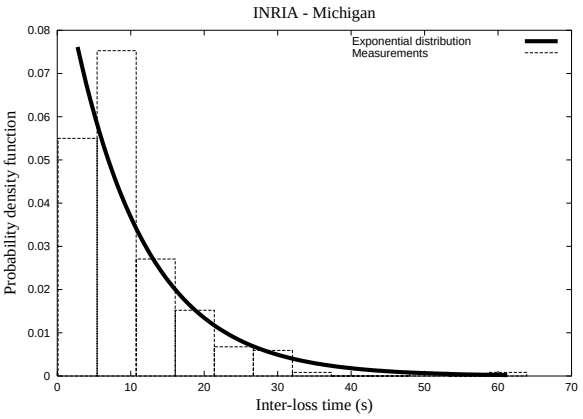


Figure 6.13: Inter-loss times: $M = 48$ Kbytes

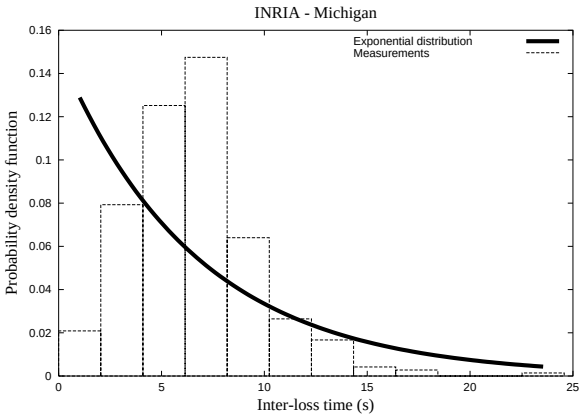


Figure 6.14: Inter-loss times: $M = 64$ Kbytes

more useful for throughput calculation.

6.9 Conclusions

We studied in this chapter TCP congestion control under the assumption that congestion events arrive according to a Poisson process, and that every congestion event results in a random number of losses or window reductions. As highlighted in [96], we reformulate the problem as an M/G/1 queuing problem with service times dependent on system workload. Using some techniques from queuing theory [80], we calculated explicit expressions for the moments of TCP rate. We profited from the analysis to also find the distribution of TCP rate and TCP window. We compared then our results to measurements from TCP connections over the Internet. Good match is reported when the loss process is close to Poisson.

Our analysis showed well the complexity of the calculation of the throughput when a limitation of TCP window exists. Even for a Poisson loss process, which the simplest process one can imagine after a deterministic loss process, the calculation is quite difficult and involves multiple constants and multiple infinite series. This justifies the use of approximations of TCP throughput we introduced in the previous chapters. However, one can always think about calculating exactly the throughput of TCP for more complicated loss processes that are based on exponential distributions as a Markov Modulated Poisson Process (MMPP) [59], or even a Markovian Arrival Process (MAP) [20]. In addition to TCP, this calculation will be certainly useful for some state-dependent queueing systems. This is one of our future directions for improving the modeling of TCP congestion control, but it seems not so urgent given the actual trend to adapt dynamically the receiver window with the congestion window [118], and given the introduction of the window scale option [77] that will permit a TCP receiver to advertise large windows up to 2 Gbytes.

Appendix

We shall prove here that the infinite series in Equation (6.24) converges. Denote $a_i = c_i/c_0$. Using (6.23), a_i satisfies the following recurrent relation

$$a_i = \frac{\nu^i}{\nu^i - 1} \sum_{k=1}^{i} q_k a_{i-k}.$$

To prove that series $\sum_i a_i$ is absolutely convergent, it is enough to prove that the majorant series $\sum_i b_i$, with b_i defined below, is convergent.

$$b_i = \frac{\nu^i}{1 - \nu^i} \sum_{k=0}^{i-1} b_k.$$

Consider b_{i+1},

$$b_{i+1} = \frac{\nu^{i+1}}{1 - \nu^{i+1}} \sum_{k=0}^{i} b_k = \frac{\nu^{i+1}}{1 - \nu^{i+1}} \left(\frac{1 - \nu^i}{\nu^i} b_i + b_i \right) = \frac{\nu}{1 - \nu^{i+1}} b_i.$$

Thus, $b_{i+1}/b_i \to \nu$ as $i \to \infty$, and therefore, the series $\sum_i b_i$ — and hence the infinite series in (6.24) — is absolutely convergent for $\nu < 1$.

Chapter 7

TCP congestion control and large bandwidth-delay product networks

Until now, we studied the performance of TCP congestion control independently of the network type. The loss process seen by the connection was only used to characterize the path between the source and the destination. By a loss we mean a congestion event that results in the reduction of TCP window by a constant factor $\nu \in (0, 1)$, typically $\nu = 0.5$. Our focus was mainly on the additive-increase multiplicative-decrease policy of the protocol. Given a certain loss process, our previous results can be used to improve the congestion control policy of TCP. For example, one can change the window increase rate during congestion avoidance so as to improve the fairness of the protocol (e.g., see [60]). But, as we conclude from the overview we presented in Chapter 2, the performance of TCP cannot always be improved on end-to-end basis. Some particular mechanisms are required to help the protocol in some challenging environments. These mechanisms are located within the network and sometimes they require modifications at the source (e.g., ELN [53], ECN [61]). In principle, these mechanisms should result in a lower loss rate or a smaller round-trip time, and thus in a better performance of TCP transfers. The performance of TCP in presence of such mechanisms require a particular study. This is what we called in the introduction the network specific approach. We have to evaluate the impact of such mechanisms on the reaction of the network (e.g., the way with which packets are dropped) in order to evaluate their impact of the performance of the protocol. One can see the difference between the end-to-end approach and the network specific approach as follows. The first approach studies the throughput of TCP for a given loss process. The second approach deals with the characterization of the loss process that the connection will see in a particular environment.

From now, we only focus on the network specific approach. We consider the three network types that we consider as the most challenging for TCP: the large bandwidth-delay product network, the asymmetric network and the wireless network. For each network type, we will explain the major problem of TCP and we develop an analytical model for the study of its performance. Based on the analysis, we will propose schemes and guidelines for the improvement of the performance of the protocol in these three environments.

We start our study by large bandwidth-delay product networks. The two other network types will be considered in the next two chapters. As we will notice later, our work on large bandwidth-delay product networks is especially useful on paths where the product is large due to a long propagation delay (e.g., a GEO satellite link where the one hop propagation delay is of about 250 ms). The major problem of TCP on such paths is the slow start phase [3, 7]. Slow start is a transitory phase designed to fill as quick as possible the network capacity at the beginning of the connection or after a timeout [75, 121]. As we shall see later, the time required to finish the slow start phase is proportional to the round-trip time of the connection and to the logarithm of the bandwidth-delay product. For the dependency of slow start duration on the bandwidth-delay product to hold, we need to assume that the slow start threshold is in the same order of magnitude as the bandwidth-delay product. Thus, in large bandwidth-delay product networks and especially those of long propagation delay, the slow start phase takes a long time which results in a poor performance of TCP transfers given the small size of the window and the low bandwidth utilization during slow start. Of course, this holds for finite transfers (some hundreds of Kbytes) for which most of the packets are transmitted during slow start. An intuitive question that one may ask here is why not to accelerate the window increase during slow start? In fact, due to the absence of any kind of packet spacing in current versions of TCP, a fast window increase results in large burstiness. This is because a TCP source transmits as many packets as its window allows upon the receipt of an ACK. A large burstiness results in packet losses and deterioration of the performance. For example, with the standard window increase policy during slow start (an increment of the congestion window by one packet upon the receipt of a non-duplicate ACK [75, 121]) and with a receiver that acknowledges every data packet, it is known [18, 84] that a TCP source injects packets into the network at twice the bottleneck bandwidth. The transmission rate of TCP will be higher with faster window increase policies. In fact, the standard window increase policy was considered when designed as a good compromise between duration of the slow start phase and burstiness of the protocol. This worked well until the appearance of large bandwidth-delay product networks which require as we said a faster slow start phase in order to use efficiently the bandwidth they propose. A faster slow start might be possible if there are enough buffers in network routers to absorb the resulting bursts. On paths where the buffers in network routers are not well dimensioned, one should expect that the acceleration of the slow start phase will result in a worse performance since packets will be dropped before filling the pipe between the source and the destination. This is the phenomenon we called *early buffer overflow* in [29]. It is an overflow that appears before what we expect when TCP is assumed to be non-bursty. An early buffer overflow results in a underestimation of the network capacity and a switch to the congestion avoidance phase at a small window. Given the slow window increase during congestion avoidance, this results in a low transmission rate and a performance deterioration.

The problem of early buffer overflow during slow start has been first studied in [18, 35, 84]

and this is for the standard version of slow start and for very small buffers in network routers. For example, it has been shown that when the buffer size in the bottleneck router is less than one third the bandwidth-delay product, the slow start phase of a TCP-Tahoe connection [56] presents losses during slow start which deteriorates the throughput. The authors showed that on such paths, TCP is unable to reach without losses the slow start threshold which was assumed to be equal to half the network capacity. In [28], we generalized their study to the case of multiple routers in tandem. We showed in that work that the problem of early buffer overflow still exists but it is less important since the burstiness of TCP is absorbed by multiple routers not only by the bottleneck one. The situation could be however worse if the buffers in the routers located between the source and the bottleneck are not well provisioned. We presented guidelines for how to dimension the buffers in network routers so that to absorb the burstiness of the standard version of slow start. Our work in [28] could be simply extended to the other window increase policies.

The slow start phase of TCP presents another problem. When the slow start threshold is set to less than the network capacity, which is very probable at the beginning of the connection where a default value is given to this threshold [10], the network gets into congestion before the end of slow start and some packets from the connection are dropped. Normally, this should result in a correction of the slow start threshold and a quick resumption of the transmission in the congestion avoidance mode at the newly estimated rate. However, it has been shown in [72] that due to the fast window increase during slow start, many packets are dropped from the same window of the connection which results in a failure of Fast Recovery, a timeout and a new slow start. Given the coarse granularity of TCP retransmission timer (multiple of 500 ms) and the time taken by slow start, the result is a performance deterioration. The author in [72] has proposed to help TCP in the estimation of a more accurate value for the slow start threshold so that the source switches to the congestion avoidance mode before the congestion of the network. The proposition was to use the flow of ACKs circulating at the beginning of the connection to estimate the bottleneck rate and the round-trip time and to set the slow start to their product. Clearly, this improves the performance but the buffers in network routers must be at least able to absorb the bursts of TCP until the congestion window reaches this product (assuming that the bottleneck rate and the round-trip time are correctly estimated). In case of small buffers, the congestion of the network will not be avoided with this value of slow start threshold and the performance will not improve. A smaller slow start threshold is required in this case if we really want to avoid the congestion.

Given these different problems of the slow start phase, we dedicate this chapter to the evaluation of its performance. We refer to [28, 29, 33, 35] for more details on our work in this direction. In this chapter, we ask ourselves different questions that we try to answer via a simple analytical model. The first question is, given a certain slow start threshold and a certain buffering capacity in the bottleneck router, what is the window increase rate that the slow start phase

must implement in order to achieve its objectives. Recall that the objective of slow start is to reach quickly and without losses the congestion avoidance mode and this in the case the slow start threshold is set to less than the network capacity. When the slow start threshold is set to larger than the network capacity, the objective of slow start is to fill quickly the pipe between the source and the destination. Losses are unavoidable in this latter case but they have to appear when the network pipe is filled. We then ask the questions of how to set the slow start threshold and how to dimension the buffers in network routers so that to absorb the bursts of slow start. Finally, we discuss the current window increase policy used by TCP during slow start, that is the use of the same window increment during all the phase. Our results show clearly that this policy is not the optimal one since at the beginning of slow start the burstiness of TCP is less important than at the end. These results will motivate us to propose a new algorithm for the window increase where the increment is decreased with the congestion window. This new policy reduces the duration of slow start without adding to the burstiness of the protocol. Our different results will be validated with simulations. Note here that even though our study focuses on the slow start phase at the beginning of the connection, it can be applied to the other slow start phases as well. In particular, our study can be applied to the Tahoe case considered in [18, 84].

Before starting the analysis, we address briefly another possible and promising direction for improving the slow start phase. We mentioned that the major problem of slow start is the burstiness of packets which prohibits the use of a faster window increase policy. But, if one succeeds to pace packets during this phase at an appropriate rate, the problem of burstiness will be solved and the source can go faster in increasing its window. Some works in this direction [19, 54, 125] propose to estimate the slow start threshold (W_{th}) and the round-trip time (RTT), and then to skip the slow start phase by sending W_{th} packets at a rate equal to W_{th}/RTT. The slow start threshold can be estimated by using either the flow of ACKs or the history of the connection. This has been shown to improve considerably the performance. The challenge with these proposals is how to estimate correctly the bottleneck rate and the round-trip time so as not to overwhelm the network and the other users. The other works in this direction [1, 35] propose to pace packets while keeping the current window increase policy. Packets are paced at a rate calculated from the current window size and the round-trip time. This solves the problem of burstiness of TCP and should result in a better performance. However, the simulation study we conducted in this direction shows that this is not the case if the slow start threshold is set to larger than the network capacity. Indeed, we noticed that a TCP connection pacing its packets during slow start is more harmful to other connections at the moment of congestion than a connection sending packets in bursts. The first connection keeps the network congested during a complete round-trip time which results in drops from most of the connections and a large decrease in utilization. The second connection keeps the network congested during the time its bursts reach the bottleneck which protects the other connections from reducing their windows. In other words, a bursty connection takes alone the result of the congestion it creates in the

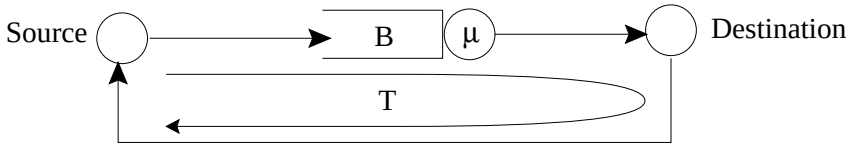


Figure 7.1: Single-node network model

network during its slow start phase, whereas a non-bursty connection makes the others also pay for its congestion. An estimation of the slow start threshold presents a solution to the problems of this second approach. The question that one may ask here is why to do slow start if we have a correct estimate of the network capacity and why not to skip this phase as with the first approach. Another question that one has to ask when deciding to pace packets is how complex is the implementation of fine-grained timers required for spacing packets. This implementation may introduce much overhead given that every expiration of a timer means an interruption of the operating system.

7.1 The model

To analyze the slow start phase of TCP, we consider a single-node model for the network between the source and the destination (Figure 7.1). This simple model for the network has been often used in the literature for the study of TCP performance [2, 18, 35, 82, 84]. The node of the model represents the bottleneck router on the path of the connection. This node has a buffer B (packets) and a rate μ (packets/s). Let T (seconds) denote the two way propagation between the source and the destination. Such network presents a bandwidth-delay product equal to μT (packets) and a pipe size equal to $B + \mu T$ (packets). Consider for the moment that the TCP connection is running alone in the network. Later, we will study the case of multiple concurrent connections. The bottleneck router is assumed to implement the classical drop-tail policy. We further consider that a file of size S is to be transfered on the TCP connection and we focus on the slow start phase at the beginning of the transfer.

The key point in the study of the slow start phase is the calculation of the window at which packets are dropped during slow start assuming that the slow start threshold W_{th} is set to an infinite value. We call this window *the overflow window* and we denote it by W_B. The calculation of W_B provides the answers to our questions. First, it tells us how the slow start threshold has to be set at the beginning of the connection in order to avoid losses. W_{th} should be set to less than W_B if we want to get in congestion avoidance before the occurrence of losses. Second, in the case W_{th} is set to more than W_B, this means in the case a congestion occurs during slow start, W_B gives us the window at which TCP gets in the congestion avoidance mode. This latter window, which is equal to the updated slow start threshold and which we denote by W'_{th}, is a direct function of W_B. Third, W_B tells us how the buffer in the network router and the window

increase rate have to be chosen. We always have to ensure that the overflow window is larger than the minimum of the slow start threshold and the pipe size. All these issues will be addressed in the following sections.

Usually, W_B is equal to the pipe size which is implicitly assumed in different works on the slow start phase (e.g., [2, 72]). But, due to the burstiness of TCP, the buffer in the bottleneck router may fail to absorb the bursts of slow start resulting in an early buffer overflow and a W_B less than the pipe size. In [18, 35, 84], the authors found the expression of W_B in case of a small buffer B. However, in their model, neither the bandwidth-delay product nor the window increase rate were considered. In this chapter, we will find the general expression of W_B. This expression will help us to understand the different interactions between the buffering capacity in the bottleneck router, the bandwidth-delay product and the window increase policy at the TCP source. We will see that W_B is upper bounded by the pipe size and that it is a decreasing function of the window increase rate during slow start and of the buffer size in the bottleneck router.

7.1.1 A model for TCP during slow start

Let $W(t)$ denote the window size in packets at time t. The transfer is assumed to start at time 0. We focus on the slow start phase at the beginning of the transfer. Suppose that after one round-trip time, the window increases by $W(t)/d$ packets where d is a positive non-null real number. With this factor d, we are able to model the window increase rate during slow start. Clearly, this rate is a decreasing function of d. The factor d can be the result of the receiver delaying ACKs and sending an ACK every d packets, and of the sender increasing its window by one packet for every non-duplicate ACK (standard slow start algorithm [121]). For example, $d = 1$ means that the receiver is acknowledging all packets. $d = 2$ represents the delay ACK mechanism widely implemented in TCP receivers [121]. Note that when we delay ACKs at the receiver, we impact both the slow start phase and the congestion avoidance phase. Without an explicit notification from the source, it is not possible to distinguish between the two phases at the receiver in order to adopt different acknowledging strategies. Another way to implement d is to change the window increase policy at the TCP source (by changing the window increment) while keeping the receiver unchanged. This will permit us to accelerate (reduce d) or to slow (increase d) the window growth during slow start without impacting the congestion avoidance phase. For example, in presence of the delay ACK mechanism, the standard version of slow start corresponds to $d = 2$ since the window is multiplied approximately by 1.5 every round-trip time. Another example is the Byte Counting solution [2] which proposes to accelerate the window increase at the source so that d passes from 2 to 1. The objective is to recover from the slowness caused by the delay ACK mechanism. As its name indicates, Byte Counting is implemented by accounting for the number of bytes acknowledged rather than the number of ACKs received. One can envisage the study of other values of d. This is what we will do in this chapter.

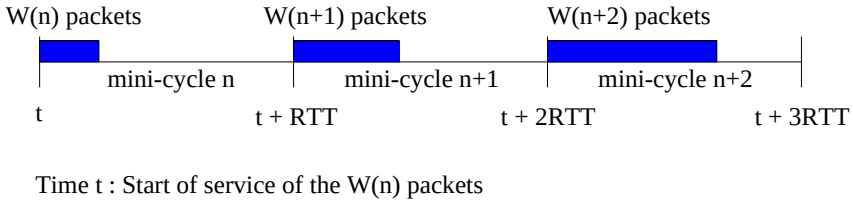


Figure 7.2: Bursts at the output of the bottleneck

7.1.2 The overflow window W_B

As in [18, 84], we divide slow start into mini-cycles. The duration of a mini-cycle is equal to one round-trip time. The first mini-cycle starts with the first packet transmitted over the connection. The word mini-cycle is used to distinguish these small cycles from the large TCP cycle between two congestion events. Let $W(n)$ be the window size at the end of mini-cycle n. It is also equal to the number of packets transmitted during mini-cycle n. The next mini-cycle starts when the ACK for the first packet of these $W(n)$ packets reaches the source. The window size at the end of the next mini-cycle is equal to

$$W(n+1) = W(n) + W(n)/d = \alpha W(n), \tag{7.1}$$

with

$$\alpha = (d+1)/d.$$

Suppose that the recurrent relation (7.1) is valid for all $n \geq 0$. Suppose also that the transfer starts with a window equal to one packet. Thus,

$$W(n) = \alpha^n W(0) = \alpha^n.$$

This equation shows clearly how the window increases exponentially during slow start. It also shows that for a constant round-trip time, the time required to reach W_{th} is equal to $RTT \log_\alpha(W_{th})$. This validates our claim at the beginning of this chapter that the duration of the slow start phase is proportional to the round-trip time and to the logarithm of the slow start threshold.

Due to the absence of any kind of packet spacing at the source or at the receiver, TCP packets are transmitted in long bursts with a frequency of one burst every round-trip time and with the number of packets in a burst equal to the window size at the end of the round-trip time [84, 128]. We assume that the source always has data to send. Figure 7.2 explains how TCP packets propagate in the network in long bursts. The figure shows the long bursts at the output of the bottleneck router. The rate of packets within a burst (we call it the rate of a burst in the following) is equal to μ (packets/s) at the output of the bottleneck. However, at the input of the bottleneck, this rate is higher than μ. It changes between slow start and congestion avoidance. During congestion avoidance, it is approximately equal to μ (it is equal to $\mu(W+1)/W$, see [28]

for explanation). During slow start, this rate is higher than μ and it is clearly a function of the parameter d which models the window increase rate. Note here that long bursts of packets are different than small bursts of packets transmitted by TCP upon the receipt of ACKs. We are working with a burstiness at a higher level. Long bursts of packets reaching the bottleneck router wait in B until they are served. A long burst of length $W(n)$ is served during mini-cycle n and it is followed by an idle period until the arrival of the burst of the following mini-cycle. This idle time between bursts disappears when the window exceeds the bandwidth-delay product (μT). In presence of exogenous traffic especially in the backward direction, and in presence of multiple routers on the path of the connection, this burstiness of TCP might be smaller since packets of a TCP connection might be separated by packets from other connections which spreads the packets of long bursts over all the path. It is not clear whether the burstiness of TCP completely disappears when packets are multiplexed inside the network. Simulations in [128] showed that this phenomenon of long bursts still exists in the case of multiple concurrent connections. The model and the simulations in [18] show that it also exists in case of a single bottleneck router and a TCP connection that shares the bottleneck bandwidth with a constant rate exogenous flow.

Given that the number of packets transmitted during a mini-cycle increases by a factor α (Equation (7.1)), we can suppose that the rate of packets within long bursts is equal to $\alpha\mu$ (packets/s), of course if the part of the path between the source and the bottleneck router has a higher rate than $\alpha\mu$. It is this high rate of packets within long bursts that may cause an overflow of the buffer B before the connection fills the pipe. Indeed, when a long burst reaches the bottleneck router, a queue starts to build up in B at a rate $\alpha\mu - \mu = \mu/d$ (packets/s). Two cases here must be considered. The first case is when B does not contain any packet from the previous mini-cycle. This happens when the window size in the previous mini-cycle is less than the bandwidth-delay product. The second case is when some packets from the previous mini-cycle are waiting in B. In the literature [18, 35, 84], only the first case is considered.

Consider the first case. A burst of size $B(d+1)$ packets is required to fill the buffer. Let n_B^1 be the number of the mini-cycle during which B overflows. The number of packets transmitted during this mini-cycle must be larger than $B(d+1)$ otherwise the overflow would not have occurred. But, the number of packets transmitted during the previous mini-cycle $n_B^1 - 1$ must be less than $B(d+1)$ otherwise the overflow would have occurred during this mini-cycle. Thus, n_B^1 satisfies,

$$\alpha^{n_B^1-1} < B(d+1) \leq \alpha^{n_B^1}.$$

For the first case, we also have

$$\alpha^{n_B^1-1} < \mu T.$$

According to our definition of d (Equation 7.1), the transmission of a long burst of $B(d+1)$ packets requires an increase in W by B packets from the beginning of mini-cycle n_B^1. It follows that,

$$W_B = W(n_B^1 - 1) + B = \alpha^{n_B^1-1} + B. \tag{7.2}$$

Now, we consider the second case. The window size is larger than μT packets and some packets are waiting in the buffer at the beginning of the mini-cycle. The increase in the window until the congestion is simply equal to the number of empty places in the buffer. Suppose that the overflow happens during mini-cycle n_B^2 and let us put ourselves at the beginning of the mini-cycle. Then, W_B changes and becomes equal to

$$W_B = W(n_B^2 - 1) + B - (W(n_B^2 - 1) - \mu T) = B + \mu T. \tag{7.3}$$

Two expressions for W_B are then available. If the window size during mini-cycle $n_B^1 - 1$ is less than μT packets, then W_B will be given by Equation (7.2), otherwise it will be given by Equation (7.3). We can combine these two expressions in a single one as stated in the following theorem.

Theorem 7.1 *If slow start is not terminated before the occurrence of losses, the buffer at the entry of the bottleneck link will overflow at a window*

$$W_B = B + \min\left(\mu T, \alpha^{n_B - 1}\right),$$

with n_B given by

$$\alpha^{n_B - 1} < B(d+1) \leq \alpha^{n_B}.$$

The following corollary can be directly derived.

Corollary 7.1 *The bottleneck buffer will not overflow during slow start if W_{th} is set to less than the W_B given by Theorem 7.1.*

To simplify the analysis, we approximate α^{n_B} by $B(d+1)$. The same approximation has been made in [18, 84]. It is equivalent to saying that the overflow of the buffer only happens at the end of a mini-cycle. The expression of W_B becomes,

$$W_B = B + \min\left(\mu T, Bd\right). \tag{7.4}$$

It is clear that this window is equal to the pipe size $B + \mu T$ whereas B is larger than $\mu T/d$. Once B becomes less than $\mu T/d$ (due to a small buffer or a fast window increase during slow start), slow start becomes unable to fill the pipe. This is where the problem of early buffer overflow appears. This expression of W_B also tells us that the appropriate value for the slow start threshold is a function of the buffer size. A value of W_{th} equal to the bandwidth-delay product as proposed in [72] does not always avoid the congestion. In the following, we will validate these results with simulations. We will also investigate the interaction between the buffer size and the slow start phase in the case of multiple concurrent connections.

Figure 7.3: The simulation scenario

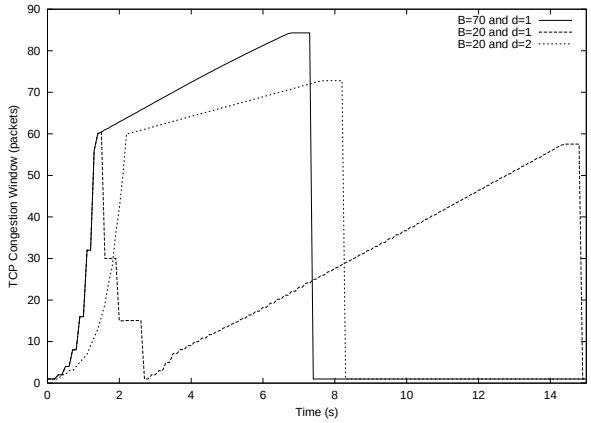


Figure 7.4: Simulation: TCP congestion window vs. time

7.2 Impact of W_{th} on the performance

As we said, the correct value for W_{th} (i.e., just less than W_B) is a function of all the model parameters not only μ and T. It decreases with the decrease in the buffer size or the increase in TCP burstiness. If the buffer size is less than $\mu T/d$, it becomes independent of the available bandwidth! For example, if we set W_{th} to the value proposed in [72] (i.e., the bandwidth-delay product), we find that a B larger than $\mu T/(d+1)$ packets is required for this value to work otherwise losses will not be avoided and the performance will not improve.

For validation, consider the simulation scenario in Figure 7.3. The Reno version of TCP is used [56]. The source transmits a file of size 100 Kbytes. TCP packets are of 512 bytes and the receiver window is set to a large value. We give two values to B: 70 packets which is approximately equal to the bandwidth-delay product and 20 packets. For a W_{th} equal to 50 packets, we plot in Figure 7.4 the congestion window as function of time. Three cases are considered: $B = 70$ packets and $d = 1$, $B = 20$ packets and $d = 1$, $B = 20$ packets and $d = 2$. We implement d by simply delaying ACKs at the receiver. Normally, for such a threshold smaller than the bandwidth-delay product, one must predict that losses will not appear during slow start. We see that it is true for $B = 70$, but it is not the case for a buffer of 20 packets and a $d = 1$. A decrease in TCP burstiness is required (from $d = 1$ to $d = 2$) to help the buffer of 20 packets to absorb the bursts of slow start and to avoid the congestion. We also notice how changing d by delaying ACKs at the receiver impacts the window growth during the congestion avoidance phase. This growth slows when d passes from 1 to 2. It would be better in the case

$B = 20, d = 2$ to keep the congestion avoidance phase unchanged.

7.3 Case of a high slow start threshold

In this section, we study the case when the slow start threshold is set at the beginning of the connection to more than W_B and when slow start ends with network congestion and losses. We consider that the factor d is implemented at the TCP source by changing the number of bytes by which the window is incremented during slow start, so that the window increase rate during congestion avoidance is not impacted by this change. Hence, the performance is a function of the window at which we get in congestion avoidance after the recovery from losses. This window, we denoted by W'_{th}, is the updated value of W_{th} or the updated source estimate of the network capacity. The smaller the W'_{th}, the longer the time required to transmit the file, of course if the file size is enough large to reach the congestion avoidance phase.

7.3.1 Calculation of W'_{th}

The buffer overflow during slow start is detected one round-trip time after its occurrence. During this round-trip time, W increases from W_B to αW_B unless the source gets in congestion avoidance. This latter case corresponds to $W_B < W_{th} < \alpha W_B$. Congestion detection then happens at a window W_D equal to

$$W_D = \min\left(W_{th}, \alpha W_B\right).$$

Once the congestion is detected, TCP sets W and W_{th} to half W_D and starts to recover from losses. Ideally, and as we described in Chapter 2, the source should recover quickly from losses and start congestion avoidance at half W_D. Unfortunately, it is far from being the case due to the large number of packets lost [72]. Even the most sophisticated versions of TCP as SACK [56] fail to recover from these losses. A timeout then occurs and the the source divides its estimate another time by two to start congestion avoidance at about $W_D/4$. The Reno version of TCP [56] presents a slightly different behavior. Reno divides its estimate of the network capacity by two upon every detection of a packet loss [16]. However, this version cannot recover from more than two packet losses in the same window without the occurrence of a timeout. Thus, with Reno, there is sometimes another division of W_D by two which results in a congestion avoidance phase that starts at $W_D/8$ (see the line corresponding to $B = 20, d = 1$ in Figure 7.4). Concerning the Tahoe version [56], it is the only one able to start congestion avoidance at $W_D/2$, but this is at the cost of a long slow start phase. Our objective here is not to compare the different versions of TCP but rather to show that the performance is a function of the overflow window W_B, which is a function of the buffer size in the bottleneck router and the aggressiveness of the slow start phase in addition to the bandwidth-delay product. In the following, we will see how

[16] This problem has been corrected by New-Reno [64] which stays in the same Fast Recovery phase for all losses in the same window.

the performance of TCP varies with d and B. We suppose for simplicity that the slow start threshold is set to a high value so that the congestion is always detected at $W_D = \alpha W_B$.

7.3.2 Interaction between buffer size and slow start aggressiveness

We use the updated network capacity estimate W'_{th} to decide on how the performance varies with d and B. When the buffer size is large enough to absorb slow start bursts, any change in d does not change the overflow window which remains equal to the pipe size (Equation (7.4)). Thus, an acceleration of the window increase improves the performance since it reduces the time taken by slow start without changing the estimate of the network capacity. This happens whenever B is larger than $\mu T/d$ packets. The problem of capacity underestimation exists when the buffer is smaller than $\mu T/d$ packets so that it overflows before the network pipe is filled. The source then gets in congestion avoidance at a small window and requires a long time to compensate the resulting capacity underestimation. This should result in a deterioration of the performance. For such small buffers, an acceleration of the window increase deteriorates further the performance since it decreases the overflow window, and hence the network capacity estimate. It is better to stop increasing the aggressiveness of slow start when B becomes less than $\mu T/d$.

To validate our result, we consider the simulation scenario in Figure 7.3 together with the two versions of TCP: the standard version (STCP) and another version that implements Byte Counting (BC) [2]. The receiver is supposed to acknowledge every other data packet. Thus, the first version corresponds to $d = 2$ and the second one to $d = 1$. We compare in Figures 7.5 and 7.6 the throughput of these two versions under two buffer sizes: 20 and 70 packets. The throughput is plotted as a function of the size of the transfered file. For a large buffer, Byte Counting works perfectly and gives better performance than standard TCP [17]. However, for a small buffer, Byte Counting is so aggressive that it fills the buffer before filling the pipe. This gives a lower network capacity estimate and thus a lower performance for most of the file sizes although standard TCP adopts a slower window increase [18].

The problem with the current window increase strategies (e.g., standard slow start, Byte Counting) is that they use the same value of d (window increment) during the entire slow start phase. But, as we saw from the analysis (Figure 7.2), the length of long bursts generated by TCP during slow start increases with the window size. At the beginning of slow start, we have smaller bursts than at the end when the window approaches the network capacity. If d is chosen in a way that the buffer in the bottleneck router absorbs all the bursts of slow start, we will get a very conservative behavior at the beginning of slow start when the window is small. We can profit from the small length of bursts when the window is small to use a lower value of d and thus to reduce the duration of slow start without changing the window at which the congestion

[17] The low performance of Byte Counting we notice in the middle of Figure 7.6 results from those files that are transfered without losses in case of standard TCP and with losses in case of Byte Counting.

[18] The low performance of standard TCP we notice for small file sizes in Figure 7.5 results from those files that are transfered without losses in case of Byte Counting.

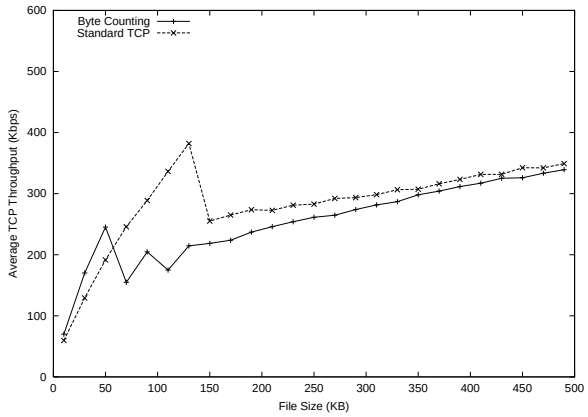


Figure 7.5: BC vs. STCP for $B = 20$

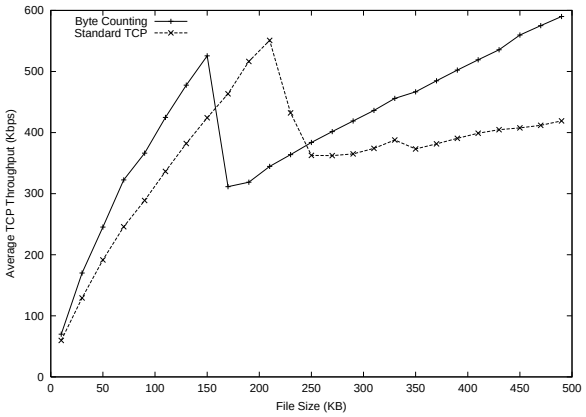


Figure 7.6: BC vs. STCP for $B = 70$

occurs. The best performance is obtained when the TCP source starts with the smallest possible value of d, then switches to a larger one just before the overflow of B and continues like this until the network pipe is filled or W_{th} is reached. This corresponds to adapting the factor d in a way that the overflow window W_B remains slightly larger than the current window instead of maintaining it constant and slightly larger than the pipe size. Note that, for a given window size and a given value of d, one can define the overflow window we calculated in Theorem 7.1 as the window at which packets will be dropped in the bottleneck router if we use the same value of d in the rest of the slow start phase.

Using (7.4), we plot in Figure 7.7 the variation of d as a function of W so that W_B is dynamic and equal to W. This is how d must be changed to get the best performance. We suppose that d cannot be taken less than a certain value d_0 (i.e., this corresponds to a maximum limit on the window increase rate). The figure has not a meaning for W larger than μT since there is no constraint on the value of d in this region. For any window W less than μT, if we use a d located above the curve, we are sure that the overflow window for this point (W, d) is larger than W and thus the buffer in the bottleneck router is able to absorb the bursts generated by TCP at this window size. If we take a W below the curve, this means that W_B is less than W and the buffer overflows at this window. We have to change the value of d in a way to be always above the curve, and the best performance is obtained when we follow the curve.

Suppose that we know a value of d that permits slow start to fill the network pipe. Call this value d_{max}. It is clear from the figure how using this value of d during all the slow start phase is very conservative, and how the performance can be improved by using any line in the conservative region located below d_{max}. It is not clear for the moment how to define such a line since this requires a knowledge of the buffer size in the bottleneck router. Using the slow start threshold, one can envisage the use of a straight line that goes from d_0 to d_{max}. This is the idea behind the *Decreasing Byte Counting* solution we will study in the next section.

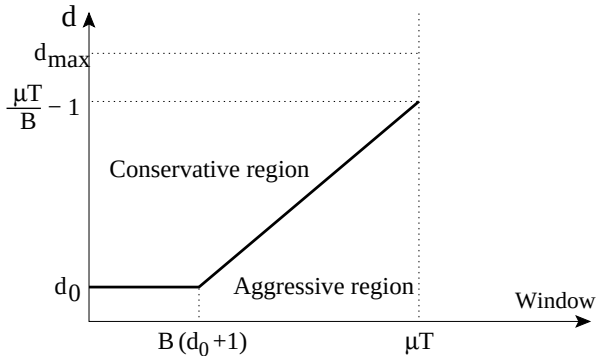


Figure 7.7: Adaptation of d

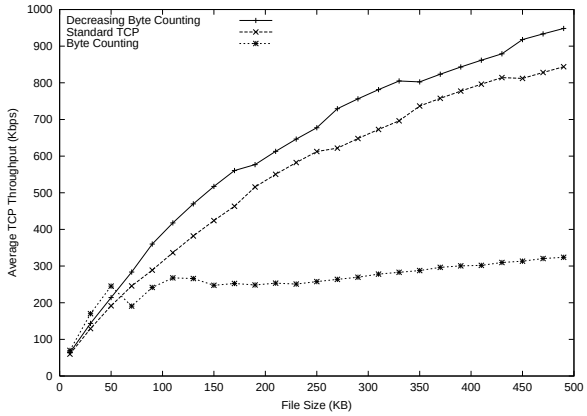


Figure 7.8: Performance of DBC

7.4 Decreasing Byte Counting

We said that maintaining the same value of d during slow start is not the optimal solution since at the beginning of slow start the problem of burstiness is not as pronounced as at the end. It would be better to use a small d at the beginning of slow start before using a large d at the end. This should reduce the duration of slow start without changing the overflow window which improves the performance. We also said that one of the possible variations of d is to increase it linearly from the small value to the large value. We validate all these conclusions using the two versions of slow start: the standard one ($d = 2$) and the Byte Counting one ($d = 1$). The receiver is supposed to acknowledge every other data packet. It is clear that Byte Counting gives better performance than standard TCP when the buffer is large. We focus on the region where Byte Counting is aggressive and where standard TCP is not. By combining these two versions, we try to propose a new slow start version that gives better performance than both of them in this region.

Based on our analysis, we propose to increase d linearly from 1 to 2 during slow start. This is equivalent to applying Byte Counting at the beginning of slow start then in starting to get out of Byte Counting towards standard TCP as long as W grows. We call such behavior Decreasing Byte Counting [29]. To realize this linear increase in d as a function of W, we use the slow start threshold as the target window for which d is equal to 2. The slow start threshold is assumed to be less than the network capacity. Starting from $d = 1$ for $W = 1$, we increase d as follows

$$d(W) = 1 + \frac{W - 1}{W_{th} - 1}.$$

The solution can be simply implemented by incrementing the congestion window upon the receipt of a non-duplicate ACK by

$$W = W + \frac{2}{d(W)}$$

instead of $W = W + 1$ in case of standard TCP and $W = W + 2$ in case of Byte Counting. Note that if we take the slow start threshold very large, Decreasing Byte Counting converges to Byte Counting since d remains close to one. If W_{th} is set to less than the network capacity, and if the buffer in the bottleneck router is able to absorb the bursts of standard TCP, Decreasing Byte Counting will reduce the duration of slow start compared to standard TCP without adding to the burstiness of the protocol. This should result in a better performance than the two other versions of TCP. Now, if the buffer in the bottleneck router is not able to absorb the bursts of standard slow start, clearly it will not be able to absorb the bursts of our version. Decreasing Byte Counting and standard TCP become aggressive approximately for the same buffer B at the bottleneck. Thus, we come with a new version of slow start that is able to reduce the duration of the phase without adding to the burstiness. Of course, this requires that the version of slow start that uses a constant d is not aggressive and that the slow start threshold is set to less than the network capacity. If it is not the case especially for the condition on the slow start threshold, our solution might be a little more aggressive. We don't pretend that our solution always improves the performance. We just want to show that keeping the value of d constant during the slow start phase is suboptimal.

We validate our version of slow start using a `ns` simulation. We show in Figure 7.8 a comparison between Byte Counting, standard slow start and our version. W_{th} is set to the bandwidth-delay product. The simulation scenario is the same as that in Figure 7.3. A buffer size of 30 packets is used in the bottleneck router. With such a buffer, Byte Counting is unable to reach the bandwidth-delay product whereas standard slow start is. We see clearly how Byte Counting causes losses and reduces the performance with respect to standard slow start. Our version is able to increase faster the window while avoiding losses. It provides the best performance with respect to the two others.

7.5 Case of multiple TCP connections

We discuss briefly the interaction between slow start and buffer size when multiple TCP connections share the bandwidth of the bottleneck router. We will see how the problem of slow start burstiness becomes less important and how the same buffer is able to support burstier slow start versions. Also, we validate in this context the benefits of our Decreasing Byte Counting algorithm.

7.5.1 A model for the case of multiple connections

Suppose that a new connection arrives at a random time to a network with already running connections. Suppose for simplicity that all the connections including the new one have approximately the same round-trip time. Consider the single-node model for the network where all the connections share the bandwidth μ (packets/s). Given that the load in a TCP/IP network is at

most divided by two, the new connection will see in the network a number of packets between half the pipe size and the pipe size. Call N the number of packets it finds. Using N, we will try to find the parameters of the equivalent single-node network seen by the new connection. Once these parameters are calculated, we can apply our previous analysis to characterize the slow start phase of the new connection.

If N is smaller than μT, we consider that the new connection sees an empty buffer together with N packets propagating along network links (i.e., not waiting for transmission in the buffer). The other connections are assumed to operate in congestion avoidance where no queue builds up in B until N exceeds the bandwidth-delay product [84]. The packets of the new connection are also assumed not to arrive simultaneously at the bottleneck router with packets from other connections. In our simulations, we noticed that this is indeed the case when the round-trip times of the different connections are approximately equal and this is simply due to the phenomenon of long bursts we already described. One can think about using the model in [18] for the case when packets of the new connection arrive simultaneously at the bottleneck router with packets from other connections. Based on our assumptions, the equivalent single-node network seen by the new connection is formed in the case $N < \mu T$ of a buffer B and a bandwidth-delay product equal to $\mu T - N$ (packets).

Now, if N is larger than μT (packets), we consider that the new connection sees full links together with $N - \mu T$ packets waiting for transmission in B. In this case, the equivalent single-node network is only formed of a buffer of size $B + \mu T - N$ packets. Thus, the overflow window given in Equation (7.4) for the single connection case can be rewritten in the case of multiple connections as,

$$W_B = B + \min\left(\mu T - N, Bd\right). \tag{7.5}$$

This new overflow window takes its value between zero and a maximum value we call W_B^{max} which corresponds to a number of packets in the network equal to $N = (B + \mu T)/2$. It follows,

$$W_B^{max} = B + \min((\mu T - B)/2, Bd). \tag{7.6}$$

In addition to the bandwidth-delay product, the overflow window is again a function of the buffer size in the bottleneck router and of the window increase rate during slow start. The window at which the new connection gets in congestion avoidance is a decreasing function of B and tends to zero when B does. We also notice that the impact of d is less important in the case of multiple connections than in the case of a single connection. Indeed, in the case of multiple connections, the equivalent network seen by the new connection has the same buffer size (B) as the real network but a smaller bandwidth-delay product ($\mu T - N$). This reduces the length of bursts that reach the buffer. The real bandwidth-delay product is shared by the different connections whereas the buffer can be considered as only dedicated to the new connection.

Using (7.5), an early buffer overflow occurs (i.e., slow start becomes aggressive) when $Bd < \mu T - N$. It never occurs when the buffer size satisfies $Bd > \mu T - N$ with N equal to $(B + \mu T)/2$.

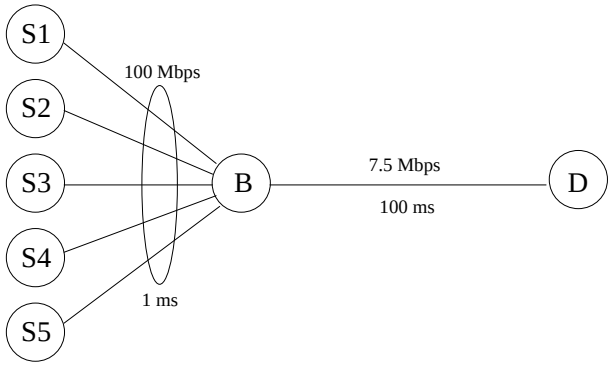


Figure 7.9: The simulation scenario

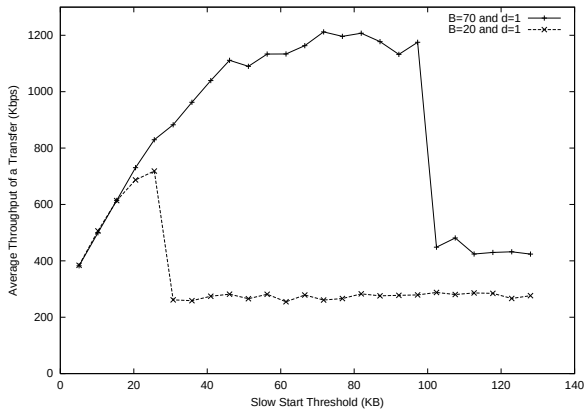


Figure 7.10: Throughput vs. W_{th}

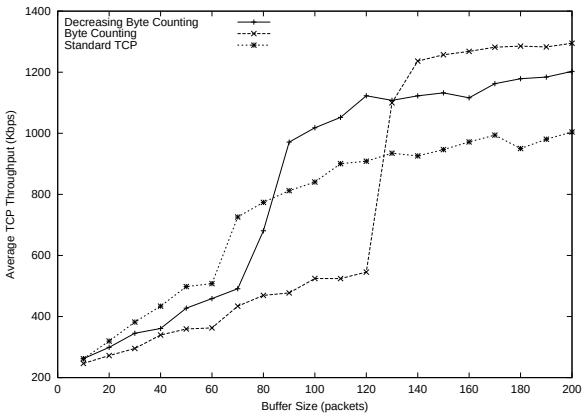


Figure 7.11: Throughput vs. buffer

This latter condition corresponds to a buffer size larger than $\mu T/(2d+1)$ (packets) which is less important than the buffer size $\mu T/d$ (packets) required in the case of a single connection. Now, even when the buffer size is smaller than $\mu T/(2d+1)$, the problem does not always occur. For example, it does not occur when N is larger to μT whatever is the size of B.

To show this behavior in presence of multiple connections, we simulate with `ns` the scenario in Figure 7.9. Five TCP-Reno sources share a 7.5 Mbps bottleneck link. Every source has many files to transmit. The file size is chosen randomly between 100 Kbytes and 1 Mbytes. Files of a source are transmitted on multiple non-parallel connections. These connections are separated by a random time between 0 and 5 seconds. The packet size is set to 512 bytes. We run 50 simulations of 50 seconds each then we calculate the average TCP throughput during a file transfer.

We plot in Figure 7.10 the average throughput of a transfer as a function of W_{th}. Two cases are considered: $B = 70$ packets and $d = 1$, $B = 20$ packets and $d = 1$. The results in this figure match well Equations (7.5) and (7.6). Theoretically, for these two cases, W_B^{max} is respectively

equal to 101 Kbytes [19] and 21 Kbytes. We see well that the throughput starts to deteriorate approximately in the middle between 0 and these values of W_B^{max}. The biggest decrease in the performance appears exactly at W_B^{max}. Thus, in case of multiple connections, W_{th} must be set according to Equation (7.5). However, overestimating the overflow window does not lead to an important degradation in performance as long as W_{th} is set to less than the maximum overflow window given in Equation (7.6).

7.5.2 Validation of Decreasing Byte Counting

We validate now the performance of our version for slow start in the context of multiple connections. We increase linearly the factor d from 1 for a window equal to 1 packet to 2 for a window equal to the slow start threshold. We use the simulation scenario in Figure 7.9. We change the buffer size in the bottleneck router and we plot in Figure 7.11 the average throughput during a transfer for the three slow start versions: standard slow start, Byte Counting and Decreasing Byte Counting. Our version gives better performance than standard TCP when standard TCP is not aggressive. For small buffers, they are both of them aggressive with our version a little more aggressive than standard TCP. Intuitively, for large buffers, Byte Counting gives the best performance. Recall, our objective is to show that, for a slow start version that uses a constant d and that is not aggressive, we can accelerate a little the window increase at the beginning of slow start without adding to the aggressiveness of the protocol.

7.6 Conclusions

We studied in this chapter the behavior of TCP during slow start and its impact on the performance. First, we calculated the value to which the slow start threshold must be set. This value can be independent of the network capacity and only a function of the buffer size. We then studied the impact of the window increase rate during slow start. We showed that accelerating the window increase improves the performance until network buffers become unable to absorb the bursts of slow start. Beyond this point, any increase in slow start aggressiveness deteriorates the performance. We defined the optimum window increase strategy during slow start and based on this, we presented a new algorithm for increasing the window that reduces the duration of slow start without adding to the burstiness of the protocol.

Note that the analysis in this chapter holds when TCP uses no kind of packet spacing to reduce the burstiness of slow start. As we explained, the trend today is to estimate the network capacity and to pace packets at an appropriate rate so as to skip the slow start phase without overloading the network routers. This will bring a final solution to the problem of slow start. Until that time, one can think about adapting the window increment or over-provisioning the

[19] To get this value of W_B^{max}, one has to consider the original expression of W_B in Theorem 7.1 where figures the number of the mini-cycle at which the congestion occurs.

buffers in routers. The problem with over-provisioning the buffers is that it increases the end-to-end delay especially on long delay paths as satellite paths. The use of active queue management techniques as RED (Random Early Detection) [42, 65] solves this latter problem. With these techniques, we can consider large buffers to absorb slow start bursts while keeping the queueing time on average at small values. We have to add to this the other benefits of the use of active queue management techniques as the increase in the utilization and the smoothness of the load and queue length.

Chapter 8

TCP congestion control and asymmetric networks

We consider now the second environment challenging for TCP, we mean by that networks presenting an important asymmetry between the bandwidth available in the forward direction and that available in the reverse direction. As an example of such networks we find the hybrid satellite network shown in Figure 2.7 where users download data from the Internet via a high speed satellite link (e.g., 2 Mbps) and send requests and ACKs via a slow reverse channel (e.g., a dial-up modem line at 64 Kbps). As we explained in Chapter 2, TCP suffers in such environment because the slow channel on the reverse path is not able to carry the flow of ACKs generated by TCP receivers. A queue of ACKs builds up in the buffer at the input of the reverse channel, we call it the *reverse buffer* in the sequel, causing an increase in the round-trip time and an overflow of the buffer (i.e., loss of ACKs). We recall briefly the different reasons for the deterioration of TCP performance. First, the increase in round-trip time caused by the queue of ACKs reduces the throughput of TCP since the transmission rate of a TCP connection is equal at any moment to the window size divided by the round-trip time [20]. Second, the increase in round-trip time as well as the loss of ACKs slow the window increase. Third, the loss of ACKs leads to burstiness at the source which may overload network buffers and cause the loss of packets. Fourth, the loss of ACKs reduces the capacity of TCP (especially Reno) to recover from losses without a timeout [56]. There is also the problem of deadlock of a new connection sharing the reverse channel with already running connections [85]. Due to the overflow of the reverse buffer, the new connection suffers from the loss of its first ACKs which prohibits it from increasing its window. This deadlock continues until the dominant connections reduce their rates. The problem of deadlock is the result of an unfairness in the distribution of the slow channel bandwidth between the different flows of ACKs. The main reason for such unfairness is that a flow of ACKs is not responsive to drops as a TCP data flow, so the running connections do not back off their rates when the reverse buffer drops some of their ACKs, and hence they do not leave some of the

[20] A TCP connection is not allowed to transmit more than a window size of packets every round-trip time. So the longer the round-trip time, the lower the transmission rate.

bandwidth they are consuming to the newly arriving connection. The running connections only reduce their rates when a data packet is lost in the forward direction or when they reach the end of the data they have to transmit.

In Chapter 2, we explain the different solutions proposed to alleviate the congestion at the entry of the slow reverse channel. The most promising solution is the one that filters the flow of ACKs in the reverse buffer [3, 25]. The advantage of this solution is that it only requires modifications in the reverse buffer without any change to the TCP code. It profits from the cumulative nature of ACKs; an ACK can be safely substituted by a subsequent ACK carrying a larger sequence number. From ACK content point of view, there is no need for queueing ACKs in the reverse buffer. Thus, when an ACK arrives at the reverse channel, the reverse buffer is scanned for ACKs from the same connection and some (or all) of these ACKs are erased. The buffer occupancy is then maintained at low levels without any loss of information.

The intuitive question that one may ask here is how many ACKs we must authorize a TCP connection to queue in the reverse buffer. In the literature, only the case of a one ACK per-connection at a time is studied [3, 25]. When an ACK arrives at the entry of the slow channel and before being queued, the reverse buffer erases any ACK from the same connection. Clearly, this behavior optimizes the end-to-end delay and the queue length, but it ignores the fact that TCP also uses ACKs to increase its congestion window. This may not have an impact on the congestion avoidance phase where the window is in general large. However, it has certainly an impact on slow start where the window is small and needs to be increased as quick as possible to achieve good performance. The impact on slow start is due to the burstiness of TCP during this phase [18, 29, 84], see Chapters 2 and 7 for an explanation of TCP burstiness during slow start. As we will see later, the fast window increase during slow start together with the absence of any packet spacing in TCP, lead to the generation of long bursts of ACKs. ACKs may also arrive in bursts due to some compression of data packets or ACKs in the network [128]. Filtering bursts of ACKs will result in few ACKs reaching the source and then in a slow window increase. This negative impact of ACK filtering will be important on finite transfers (hundreds of Kbytes) for which most of the packets are transmitted during slow start. In particular, it will be pronounced over long delay links (e.g., satellite links) where slow start is already slow enough [34]. Authorizing some number of ACKs from a connection to be queued in the buffer before the start of filtering will have the advantage of absorbing these bursts of ACKs which will result in a faster window increase. However, this threshold of ACKs must be kept at a small value in order to limit the end-to-end delay. A certain tradeoff then appears: one must expect an improvement in the performance as the threshold increases, followed by a deterioration in the performance when it becomes large (see Figure 8.5 for an example).

We focus in this chapter on the ACK filtering solution and we try to refine it in order to absorb the bursts of ACKs while not increasing the end-to-end delay. First, we study the case when the ACK filtering threshold is set to a fixed value. From now, we mean by ACK filtering

threshold the number of ACKs that a connection can queue in the reverse buffer. With an analytical model similar to the one we developed in Chapter 7, we show how this threshold must be chosen. We then present an algorithm, we call *Delayed ACK Filtering*, that adapts the filtering of ACKs as a function of the slow channel utilization rather than the ACK queue length. This is equivalent to a dynamic setting of the ACK filtering threshold. Our objective is to pass as many ACKs as possible to the source while maintaining the end-to-end delay at small values. In case of multiple concurrent connections, our algorithm adapts the filtering in a way to share fairly the slow channel bandwidth between the different connections. This provides a solution to the problem of deadlock caused by the non-responsiveness of the ACK flows. The different results of our study are validated with simulations using the network simulator `ns`. The research work we present in this chapter has been published in [31].

8.1 Impact of ACK filtering threshold

Let us focus on the slow start phase since it is the most impacted by ACK filtering. We consider short transfers and we look at the slow start phase at the beginning. Our analysis can be applied to the other slow start phases as well. We start by the case of a single transfer sharing the slow reverse channel. The burstiness of ACKs caused by slow start is only considered. Our objective is to show that delaying the filtering until a certain number of ACKs get queued in the reverse buffer shortens the slow start phase and improves the performance if this number is correctly set. We assume that the routers in the forward direction are able to absorb the burstiness of traffic resulting from the filtering of ACKs. We don't address later the problem of burstiness of traffic in the forward direction since our algorithms, by passing more ACKs, reduce this burstiness compared to the classical one ACK at a time filtering strategy.

8.1.1 TCP and network models

Let μ_r be the bandwidth available on the reverse path and let T be the constant component of the round-trip time in absence of queueing delay. μ_r is measured in terms of ACKs per second. Assume that the round-trip time only increases when ACKs are queued in the reverse buffer. When no ACKs are queued, the round-trip time is taken equal to T. A queue of ACKs starts to be always present in the reverse buffer when the reverse channel is fully utilized, this means when the number of ACKs arriving at the reverse buffer per T is more than $\mu_r T$. Our assumption on the round-trip time holds given the considerable slowness of the reverse channel compared to the other links on the path.

Assume for the moment that the reverse buffer is large and that ACKs are not filtered. The window at the TCP source then grows exponentially with a rate function of the frequency with which the receiver acknowledges data packets. We look on the slow start phase and we consider that the source applies the standard window increase of one packet per ACK [75]. Suppose that

the receiver acknowledges every d packets (usually $d = 2$), thus the window increases by a factor $\alpha = 1 + 1/d$ every round-trip time. If we denote by $W(n)$ the congestion window size at the end of the nth round-trip time, we get

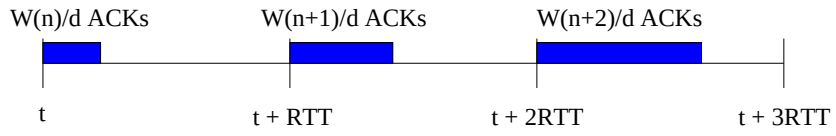
$$W(n+1) = (d+1)W(n)/d = \alpha W(n).$$

For $W(0) = 1$, this gives the following exponential expression of the window we have already seen

$$W(n) = \alpha^n.$$

Once the reverse channel is fully utilized, ACKs start to arrive at the source at a constant rate μ_r. Here, the window together with the round-trip time start to increase linearly with time. The transmission rate of TCP, which is equal to the window size divided by the round-trip time, stops increasing and becomes limited by the reverse channel. This continues until ACKs start to be filtered or dropped. The round-trip time then stops increasing and the transmission rate resumes its increase with the window size (see Figure 8.4).

The first remark that we can make here is that the ACK queue length needs to be maintained at small values in order to get a small round-trip time and hence a better performance. An aggressive filtering (say one ACK per-connection) is then needed. But, due to the fast window increase during slow start and the absence of any kind of packet spacing in TCP, ACKs may arrive at the reverse buffer in separate long bursts with the rate of ACKs within a burst higher than μ_r and with the total rate of ACKs lower than μ_r. These long bursts arrive with a frequency of one long burst per round-trip time and they are the results of the long bursts of packets that propagate in the forward direction. We refer to Chapter 7 for a description of the burstiness phenomenon during slow start. For instance, let us look at Figure 8.1 to understand how ACKs arrive at the reverse channel in long bursts. An aggressive filtering reduces the number of ACKs reaching the source whereas the long bursts of ACKs can be absorbed without causing any increase in the round-trip time. Such absorption will result in more ACKs reaching the source and hence in a faster window increase. Given that the round-trip time remains constant whenever the reverse channel is not fully utilized, a faster window increase results in a faster transmission rate increase and hence in a better performance. Thus, the general guideline for ACK filtering is to accept all ACKs until the slow channel becomes fully utilized, then to filter them in order to limit the round-trip time. We consider first the case when a connection is allowed to queue a certain number of ACKs in the reverse buffer. This number, which we denote by δ and which we call the ACK filtering threshold, is maintained constant during the connection lifetime. We study the impact of δ on the performance and we show how it must be chosen. Later, we present algorithms that adapt ACK filtering as a function of the slow channel utilization rather than the ACK queue length. This permits a simpler implementation together with a better performance than fixing a certain number of ACKs.



Time t : Start of service of the W(n)/d ACKs at the entry of the slow reverse channel

Figure 8.1: Long bursts of ACKs as they cross the reverse channel

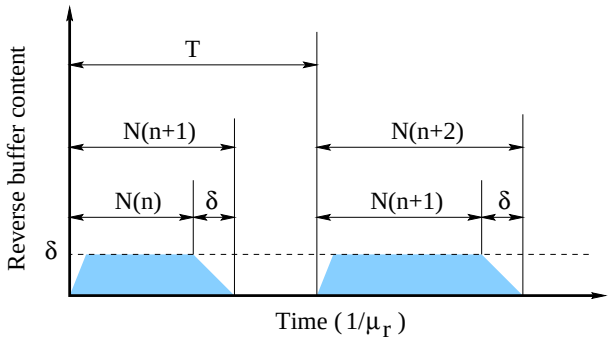


Figure 8.2: Reverse buffer occupancy as a function of time

8.1.2 ACK filtering threshold

From Chapter 7, we know that TCP transmits every round-trip time a window size of packets in a long burst. A burst of $W(n)$ packets causes the generation of $W(n)/d$ ACKs which reach the source at the end of the round-trip time. Figure 8.1 shows this bursty behavior of TCP at the input of the slow reverse channel. Given that the reverse channel is the bottleneck, ACKs within long bursts have a rate μ_r at the output of the reverse channel and a rate $\alpha\mu_r$ at its input. During the receipt of a long burst of ACKs, a queue builds up in the reverse buffer at a rate $(\alpha-1)\mu_r$ ACKs/s. A long burst of X ACKs at a rate $\alpha\mu_r$ causes the building of a queue of length $X/(d+1)$ ACKs. The full utilization of the reverse channel requires the receipt of a long burst of length $\mu_r T$ ACKs and hence the absorption of a queue of length $\mu_r T/(d+1)$. This is the value of δ_o, the optimal filtering threshold, the reverse buffer must use

$$\delta_o = \mu_r T/(d+1). \tag{8.1}$$

We notice that δ_o is an increasing function of the slow channel bandwidth and the frequency with which TCP packets are acknowledged.

8.1.3 Early ACK filtering

We consider now the case $\delta < \delta_o$. When an ACK finds δ ACKs in the reverse buffer, the lastly received ACK is erased and the new ACK is queued at its place. We call this a *static* filtering

strategy since δ is not changed during the lifetime of the connection. Let us study the impact of δ on the window increase rate, and thus on the performance of TCP.

Starting from $W(0) = 1$ (packet), the window increases exponentially until round-trip time n_0 where ACKs start to be filtered. This happens when the ACK queue length reaches δ which in turn happens when the reverse buffer receives a long burst of length $\delta(d+1)$ ACKs at a rate $\alpha\mu_r$. Recall that a long burst of X ACKs at a rate $\alpha\mu_r$ causes the building of a queue of $X/(d+1)$ ACKs. Given that the length of the long burst of ACKs received during round-trip time n_0 is equal to $W(n_0)/d$, we write

$$W(n_0) = \alpha^{n_0} = \delta d(d+1).$$

After round-trip time n_0, ACKs start to be filtered and the window increase slows. Consider a round-trip time $n > n_0$ and look at the region where the slow channel is not fully utilized. We know that the maximum window increase rate (μ_r packets per unit of time) is achieved when the reverse channel is fully utilized, and the best performance is obtained when we reach the full utilization as soon as possible. Let $N(n)$ denote the number of ACKs that leave the slow channel during round-trip time n. Given that we are in slow start, we have $W(n+1) = W(n) + N(n)$.

The long burst of data packets of length $W(n+1)$ generates $W(n+1)/d$ ACKs at the destination which reach the reverse buffer at a rate faster than μ_r. The duration of this long burst of ACKs is equal to the duration of the long burst $N(n)$ at the output of the slow channel in the previous round-trip time. Recall that we are working in a case where the bandwidth available in the forward direction is very large compared to the rate of the slow reverse channel so that many data packets can be transmitted at the source between the receipt of two ACKs. During the receipt of the $W(n+1)/d$ ACKs, a queue of δ ACKs is formed in the reverse buffer and the slow channel transmits $N(n)$ ACKs. The ACKs stored in the reverse buffer whose number is equal to δ are then transmitted. Thus,

$$N(n+1) = N(n) + \delta.$$

For explanation, Figure 8.2 shows the occupancy of the reverse buffer as a function of time after the start of ACK filtering and before the full utilization of the slow reverse channel. We can write for $n > n_0$,

$$\begin{aligned} N(n) &= N(n-1) + \delta = N(n_0) + (n-n_0)\delta \\ &= W(n_0)/d + (n-n_0)\delta = \delta(d+1+n-n_0), \end{aligned}$$

$$\begin{aligned} W(n) &= W(n-1) + N(n-1) = W(n_0) + \sum_{k=n_0}^{n-1} N(k) \\ &= \delta[d(d+1) + (n-n_0)(d+1) + (n-n_0)(n-n_0-1)/2]. \end{aligned}$$

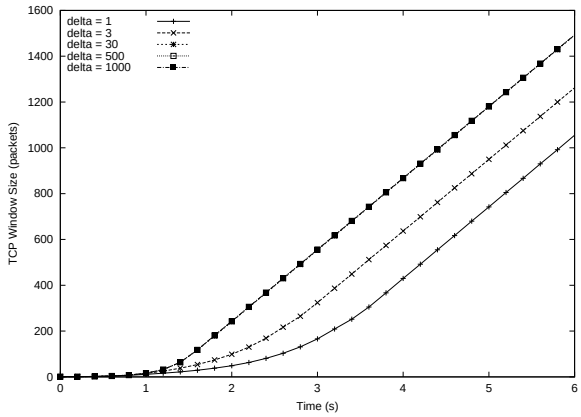


Figure 8.3: TCP window vs. time

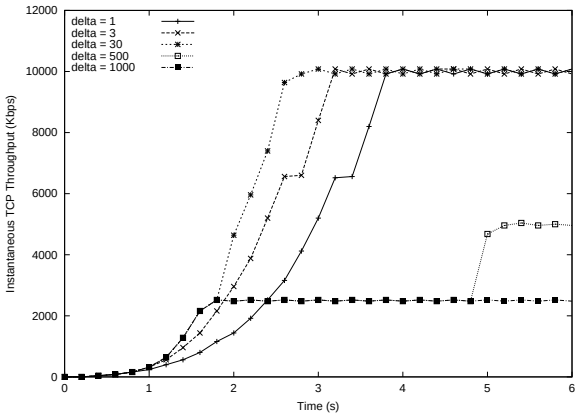


Figure 8.4: TCP rate vs. time

We remark that when taking $\delta < \delta_o$, the window increase changes from exponential to polynomial and it slows with δ. The source spends more time before reaching the maximum window increase rate of μ_r packets per unit of time, which we recall takes place when the slow channel is fully utilized. As we will see with the simulations, this results in a performance deterioration.

8.1.4 Simulation

Consider a simple simulation scenario where a TCP Reno source [56] transmits packets of size 1000 bytes over a forward link of 10 Mbps to a destination that acknowledges every data packet ($d = 1$). The forward buffer, the receiver window, as well as the slow start threshold at the beginning of the connection are set to high values. ACKs cross a slow channel of 100 Kbps back to the source. T is set to 200 ms. We use the `ns` simulator and we monitor the packets sent during the slow start phase at the beginning of the connection. We add to the simulator a static ACK filtering strategy that limits the number of ACKs in the reverse buffer to δ, and we compare the performance for different values of δ. The reverse buffer itself is set to a large value. We provide three figures where we plot as a function of time, the congestion window (Figure 8.3), the transmission rate (Figure 8.4), and the last acknowledged sequence number (Figure 8.5). The transmission rate is averaged over intervals of 200 ms.

For such simulation scenario, the calculation gives a δ_o equal to 30 ACKs (Equation (8.1)). According to our analysis, a δ less than δ_o should slow the window growth. We see this clearly for $\delta = 1$ and $\delta = 3$ in Figure 8.3. The other values of δ in the figure give the same window increase as δ_o since the filtering of ACKs starts for all of them after the full utilization of the reverse channel, hence the maximum window increase rate (μ_r packets/s) is reached at the same time. But, the observation of the window is not sufficient to study the performance of TCP since the same window means different performance if the round-trip time is not the same. In addition to the window, one has to look at the plots of the transmission rate (Figure 8.4). For

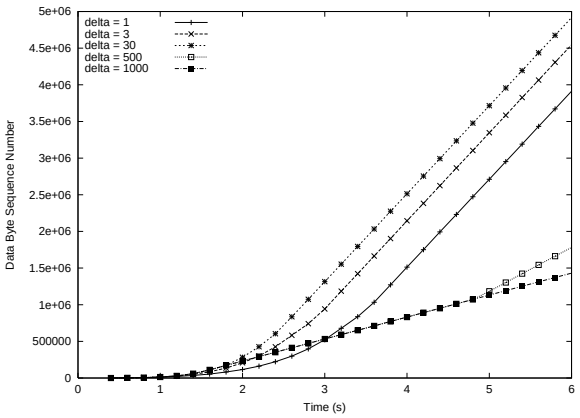


Figure 8.5: Sequence number vs. time

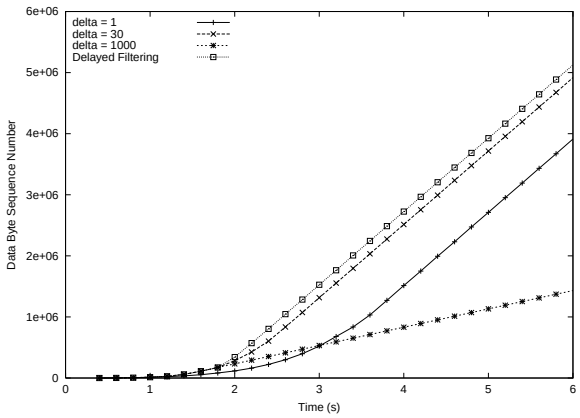


Figure 8.6: Sequence number vs. time

small δ, the round-trip time can be considered as always constant ($\simeq T$) and the curves for the transmission rate and for the window have approximately the same form. The difference is that the transmission rate saturates when it reaches the available bandwidth in the forward direction (at 1000 Kbps). We also notice in Figure 8.4 that TCP rate saturates somewhere in the middle (e.g., at 3s for $\delta = 1$). This latter saturation corresponds to the time between the full utilization of the reverse channel and the convergence of the round-trip time to its limit ($T+\delta/\mu_r$) when δ ACKs start to be always present in the reverse buffer. During the convergence time, the transmission rate remains constant due to a linear increase of both the window and the round-trip time. Once the round-trip time reaches its limit, the transmission rate resumes its increase with the window but this time linearly not exponentially. Note that the convergence of the round-trip time takes a long time for large δ (e.g., around 5s for $\delta = 500$). Now, the sequence number in Figure 8.5 is an indication of the overall performance. At any moment, the plot of the sequence number tells us how many packets have been acknowledged since the beginning of the connection. We see how taking $\delta = \delta_o$ leads to the best performance since it presents a good compromise between delaying the filtering of ACKs to improve the window increase and bringing it forward to reduce the round-trip time. While increasing δ, the overall performance improves until $\delta = \delta_o$ then worsens. Note that this behavior may not be seen with long TCP transfers where slow start has a little impact on the overall performance.

8.2 Delayed ACK filtering: Case of a single connection

Tracking the queue length for filtering ACKs is not a guarantee for good performance. First, in reality and due to the fluctuations of Internet traffic, the arrival of ACKs at the reverse buffer may be completely different than the theoretical arrival we described. Second, the calculation of the optimal threshold (Equation (8.1)) is difficult in practice since it requires the knowledge of the round-trip time and the acknowledgement strategy at TCP receivers (d). Third, setting

the filtering threshold to a fixed value leads to an unnecessary increase in round-trip time after the slow channel becomes fully utilized. Some mechanisms are required in the reverse buffer to absorb the bursts of ACKs when the slow channel is not well utilized, and to filter ACKs with a small δ otherwise. For the first and second problems, one can imagine to set the filtering threshold to the bandwidth-delay product of the return path ($\mu_r T$) in order to account for the most bursty case. The cost to pay here will be a further increase in the round-trip time.

The simplest solution to the above problems is to measure the rate of ACKs at the output of the reverse buffer — before being transmitted on the slow channel — and compare it to the slow channel bandwidth, then to use the result of the comparison to activate or halt the filtering of ACKs. Measuring the rate at the output rather than at the input is better since ACKs are more spread over time which increases the precision of the measurement tool. In case we don't know the slow channel bandwidth (e.g., case of a shared medium), one can measure how frequently ACKs are present in the reverse buffer. ACKs start to be always present in the reverse buffer when the slow channel starts to be fully utilized. When the measurement indicates that the slow channel is fully utilized (e.g., the utilization exceeds a certain threshold that we fix to 90% in our simulations), we start to apply the classical filtering studied in the literature [25]: erase all old ACKs from a connection when a new ACK arrives. Once the utilization of the slow channel drops below a certain threshold, filtering is halted until the utilization increases again. This adaptation of ACK filtering guarantees a maximum window increase during slow start and a minimum round-trip time in the steady state. We can see it as a dynamic filtering where δ is set to infinity when the slow channel is under-utilized and to 1 when it is well utilized. Also, it can be seen as a transformation of the rate of the input flow of ACKs from a given rate λ to $\min(\lambda, \mu_r)$ without the loss of any information, of course if the reverse buffer is large enough so as to absorb the bursts of ACKs until the start of filtering. Recall that we are always working in the case of a single connection. The case of multiple concurrent connections is studied later.

8.2.1 Utilization Measurement

Assume that the slow channel bandwidth is known. We use the time sliding window (TSW) algorithm defined in [52] for ACK rate measurement. When an ACK leaves the buffer, the time between this ACK and the last one is measured and the rate estimate is updated by taking a part of this new measurement and the rest from the past. The difference from classical low pass filters (i.e., exponentially weighted moving average algorithms with constant coefficients, see [75] for example) is that the contribution of the new measurement is more important at low rates than at high rates. This makes the convergence time of the algorithm independent of whether the rate of ACKs decreases or increases. Note that if the contribution of the new measurement is constant, the convergence time will only be a function of the number of measurements, and hence it will be much longer when the times between ACKs start to increase than when they start to decrease. The convergence time of the TSW algorithm is controlled via a time window

that decides how much the past is important. We can see this window as the time interval during which the rate of ACKs is averaged.

The TSW algorithm is defined as follows. Let `Rate` be the rate estimate, `Window` be the time window, `Last` be the time when the last ACK has been seen, `Now` be the current time, `Size` be the size of the ACK (40 bytes). Then, upon ACK departure,

```
1) Volume = Rate*Window + Size;
2) Time = Now - Last + Window;
3) Rate= Volume /Time;
4) Last=Now;
```

The same algorithm with small changes can be used to measure how frequently ACKs are present in the reverse buffer.

8.2.2 Simulation

We consider the same simulation scenario as that in Section 8.1.4. We add our delayed filtering algorithm to the `ns` simulator and we validate its performance. The time window is taken in the same order as the round-trip time. We plot in Figure 8.6 the sequence number of the last acknowledged packet as a function of time. We also plot in the same figure the sequence number for static filtering and three values of δ. The figure shows how our algorithm gives as good performance as the case when the threshold of static filtering is correctly chosen ($\delta = \delta_o = 30$). Moreover, our dynamic filtering slightly outperforms the optimal static filtering due to the erasing of all ACKs when the slow channel is fully utilized. The operation of delayed filtering is illustrated in Figures 8.7 and 8.8 where we plot the reverse buffer occupancy as a function of time. These figures correspond respectively to static filtering with $\delta = \delta_o = 30$ and delayed filtering. Note how bursts of ACKs are absorbed at the beginning of the connection before being filtered some time later. The filtering of ACKs starts approximately at the same time in both cases. This is the time when the slow channel becomes fully utilized. We notice how after the full utilization of the slow channel, delayed filtering reduces the ACK queue length to one whereas static filtering keeps δ ACKs in the reverse buffer.

8.3 Delayed Filtering: Case of multiple connections

Consider now the case of multiple connections running in the forward direction and sharing the slow reverse channel for their ACKs. Let N be the number of connections. In addition to the problems of end-to-end delay and reverse channel utilization, the existence of multiple connections raises the problem of fairness in sharing the reverse channel bandwidth. A new connection generating ACKs at less than its fair share from the bandwidth must be protected from other connections exceeding their fair share. This protection must be enforced by some flow isolation in the reverse buffer since as we said ACKs are not responsive to drops as data

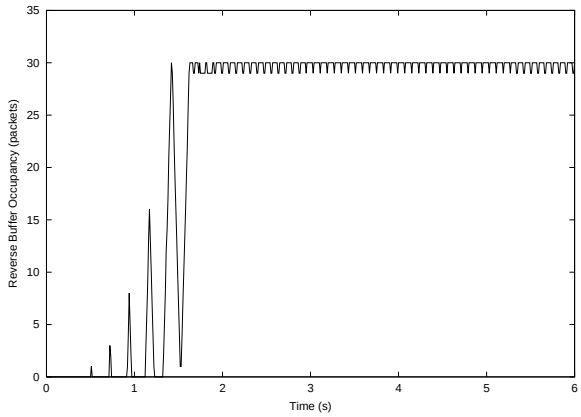


Figure 8.7: Static filtering

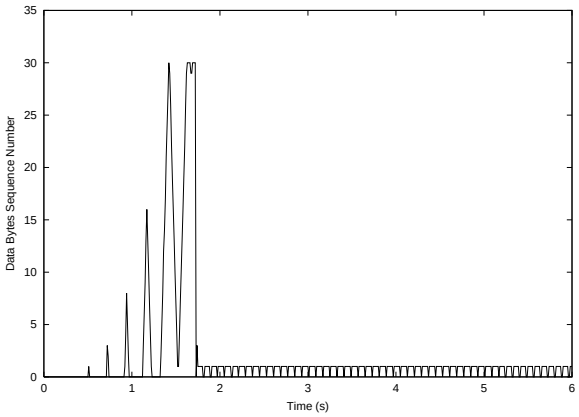


Figure 8.8: Delayed filtering

flows. We consider a max-min fairness [**?**] which consists in sharing the bandwidth of the slow channel equally between the different ACK flows. Note that other fairness schemes [**?**] could be considered. For example, one can imagine to give ACKs from a new connection more bandwidth than ACKs from an already running connection.

In this section, we present different filtering strategies and we study their performance. We consider first the case of a large reverse buffer where ACKs are not lost but queued to be transmitted later. There is no need here for an ACK dropping strategy but rather for a filtering strategy that limits the queue length, improves the utilization, and provides a good fairness. Second, we study the case when the reverse buffer is small and when ACK filtering is not enough to maintain the queue length at less than the buffer size. ACK filtering needs to be extended in this second case by an ACK dropping strategy.

8.3.1 Case of a large buffer

To guarantee a certain fairness, we apply delayed filtering to every connection. A list of active connections is maintained in the reverse buffer. For every connection, we measure and store the average rate of its ACKs at the output of the reverse buffer. When an ACK arrives at the reverse buffer, the list of active connections is checked. If no entry is found, a new entry is created for this new connection. Now, if the average rate associated to the connection of the new ACK exceeds the slow channel bandwidth divided by the number of active connections, all ACKs belonging to the same connection are filtered and the new ACK is queued at the place of the oldest ACK from the same connection. When an ACK leaves the buffer, the average rates of the different connections are updated. A TSW algorithm is again used for ACK rate measurement. The entry corresponding to a connection is freed when its average rate falls below a certain threshold.

Keeping an entry per connection seems to be the only problem with our algorithm. We believe that with the increase in processing speed, this problem does not exist. Also, and as we will see

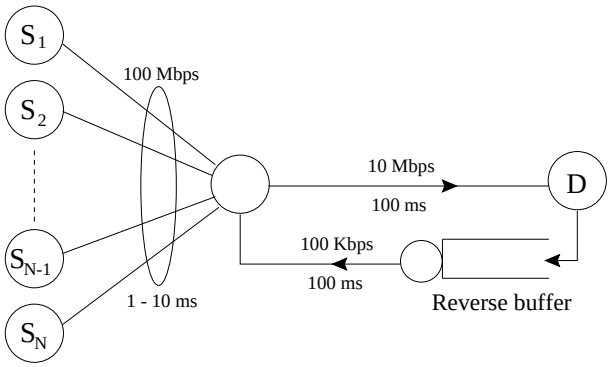


Figure 8.9: Simulation scenario

later, we can stop our algorithm beyond a certain number of connections since it converges to static filtering with $\delta = 1$. This happens when the fair bandwidth share of a connection becomes very small. Classical filtering ($\delta = 1$) could then be applied without the need to account for bandwidth sharing.

Now, delaying the filtering of ACKs from a connection separately from the other connections while keeping the classical FIFO (First-In First-Out) service does not result in a complete isolation of flows. The accepted burst of ACKs from an unfiltered connection unnecessarily increases the round-trip time of the other connections. A Round-Robin scheduling is required for the isolation to be achieved. ACKs from filtered connections will no longer have to wait after ACKs from an unfiltered one.

We add the different filtering algorithms we cited above to the `ns` simulator and we use the simulation scenario in Figure 8.9. N TCP-Reno sources transmit short files of sizes chosen randomly with a uniform distribution between 10 Kbytes and 10 Mbytes to the same destination D. The propagation delays of access links are chosen randomly with a uniform distribution between 1 and 10 ms. ACKs from all the transfers return to the sources via a 100 Kbps slow channel. A source S_i transmits a file to D, waits for a small random time, and then transmits another file. We take a large reverse buffer and we change the number of sources from 1 to 20. For every N, we run the simulations for 1000s and we calculate the average throughput during a file transfer. We plot in Figure 8.10 the performance as a function of N for five algorithms: no-filtering ($\delta = +\infty$), classical filtering ($\delta = 1$), delayed filtering with all the connections grouped into one flow, per-connection delayed filtering with FIFO and with round-robin scheduling.

In the third algorithm, we only check the utilization of the slow channel. When the utilization exceeds a certain threshold, we apply classical filtering with one ACK per-connection at a time. The filtering is halted when the slow channel is not fully utilized.

We notice in the figure that no-filtering gives the worst performance due to the long queue of ACKs in the reverse buffer. Classical filtering solves this problem but it is too aggressive so it does not give the best performance especially for small number of connections. For large number

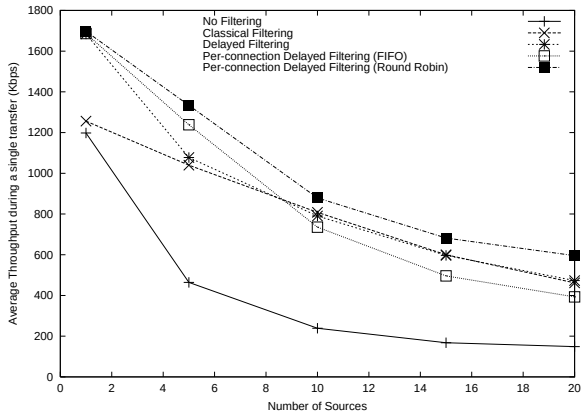


Figure 8.10: Case of a large buffer

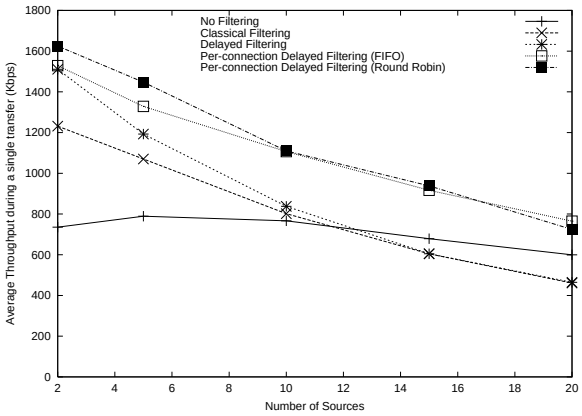


Figure 8.11: Case of a small buffer

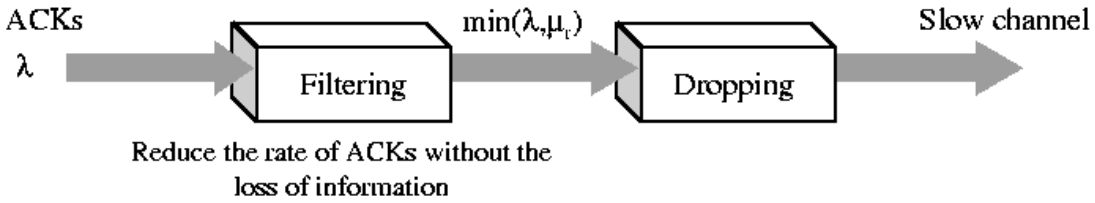


Figure 8.12: The relation between ACK filtering and ACK dropping

of connections, the bandwidth share of a connection becomes small and classical filtering gives close performance to that of the best policy, per-connection filtering with round-robin scheduling. Single-flow delayed filtering is no other than static filtering beyond a small number of connections, and per-connection filtering with FIFO scheduling gives worse performance than round robin scheduling due to the impact of unfiltered connections on the round-trip time of filtered ones. Compared to no-filtering and classical filtering, our algorithms improve the performance of TCP especially when the number of concurrent connections is not very large.

8.3.2 Case of a small buffer

The question we ask here is what happens if we decide to queue an ACK and we find that the buffer is full. In fact, this is the open problem of buffer management with a difference in our case that the flows we are managing are not responsive to drops as TCP data flows. The other difference is that in our case, ACK dropping is preceded by ACK filtering which reduces the overload of ACKs on the reverse buffer. The buffer management policy is only used in the particular case when filtering is not enough to avoid the reverse buffer overflow. We can see the relation between filtering and dropping as two consecutive boxes (Figure 8.12). The first box, which is the filtering box, tries to eliminate the unnecessary information from the flow of ACKs. The filtered flow of ACKs is then passed to the second box which contains the reverse buffer

with the appropriate dropping strategy.

For classical filtering, we use the normal drop-tail policy. The buffer space is fairly shared between the different connections (one ACK per connection) and we don't have enough information to use another more intelligent dropping policy. The same drop-tail policy is used in case of single-flow delayed filtering when ACKs are filtered. When ACKs are not filtered, we use the Longest Queue Drop policy described in [120]: the oldest ACK of the connection with the longest queue is dropped and the new ACK is queued at the end of the buffer. Now, for per-connection delayed filtering, we profit in the dropping procedure from the information on rates available for filtering: the oldest ACK of the connection with the highest rate is dropped.

We add all these dropping policies to the filtering module we developed for the `ns` simulator. We repeat the same simulation of the previous section but now with a small reverse buffer of 10 packets. The average throughput is shown in Figure 8.11 as a function of the number of sources. In this case and especially for large N, the difference in performance is mainly due to the difference in the dropping policy. This can be seen from the difference in performance at large N between per-connection delayed filtering and classical filtering. If the dropping policy is not important, these two filtering strategies should give close performance. Per-connection delayed filtering with round robin scheduling gives again the best performance. The relative position of classical filtering to no-filtering is a little surprising. When the number of connections is smaller than the buffer size, classical filtering is able to keep an empty place for a new connection and hence protects it from already running connections. This leads to a better performance than the case of no-filtering. However, as the number of connections exceeds the buffer size, the reverse buffer starts to overflow and new connections will no longer be protected. Thus, the performance of classical filtering deteriorates when N increases and it drops below that of no-filtering for large N. We can conclude that when the number of connections is larger than the buffer size, a simple drop-tail policy is enough for good performance. This again limits the number of connections that our delayed filtering algorithm needs to track.

8.4 Conclusions

We studied in this chapter the ACK filtering scheme proposed in the literature to reduce the length of the queue of ACKs on the reverse path in an asymmetric TCP/IP network. We showed that, due to the burstiness of ACKs, the existing scheme is aggressive and that the performance can be further improved by absorbing the bursts of ACKs whenever the scarce bandwidth on the reverse path is not fully utilized. We proposed two schemes for such absorption: a static scheme where the filtering starts when a certain number of ACKs is queued, and a dynamic scheme where the filtering starts when the rate of ACKs exceeds a certain threshold. Our two schemes improve the performance with respect to the classical scheme. Moreover, the schemes we proposed ensure that the scarce bandwidth is fairly shared between the different connections.

This is very important to help new connections to quickly increase their windows and hence to realize better throughputs.

The main problem with our schemes is that they require a per-connection state in the reverse buffer in order to achieve the fairness objective. But, we showed that for multiple reasons, there is no need for this per-connection state when the number of connections exceeds a certain threshold. First, when the fair share of a connection from the scarce bandwidth is small, our schemes converge to the simple filtering scheme studied in the literature. Second, when the number of connections exceeds the buffer size, the ACK dropping strategy becomes the most important factor. In this second case, one can stop filtering ACKs and only use an ACK dropping strategy (e.g., Longest Queue Drop [120]) that ensures fairness. Our results showed that even a simple drop-tail strategy gives better performance in this case than the classical ACK filtering. The dropping strategy we proposed to our per-connection scheme gives the best performance.

Chapter 9

TCP congestion control and wireless networks

The third and the last environment we consider for the study of TCP congestion control is the wireless environment. We mean by wireless environment any network containing an air interface. This ranges from terrestrial mobile networks (e.g., GSM) to GEO satellite networks. As we explained in Chapter 2, the problem of TCP in such networks is in the transmission errors on the wireless interface which are interpreted by TCP as congestion signals [7, 26, 34]. Indeed, due to many phenomena as fading, signal-power attenuation, shadowing, interference, etc., wireless links are known to have a much higher Bit Error Rate (BER) than wire lines. The Signal-to-Noise ratio at the output of a wireless link often drops to low values resulting in a corruption of the transmitted information. A TCP packet corrupted while crossing a wireless link is discarded before reaching the TCP receiver which results in a loss detection at the TCP source and an unnecessary reduction of the congestion window. In the following, we will use the term non-congestion losses to denote corrupted TCP packets on a wireless interface. We will also use the terms noisy link and lossy link to denote the wireless link. Note that even though this chapter is dedicated to the wireless environment, it is useful for any other environment where TCP packets are lost for other reasons than congestion. A typical example could be the loss of ATM cells — and hence of the TCP packets they belong to — on a UBR (Unspecified Bit Rate) virtual circuit [21, 68] connecting two adjacent IP routers.

In Chapter 2, we presented an overview of the different solutions proposed to improve the performance of TCP on paths with non-congestion losses. We saw that the most promising solution is the one that improves the quality of the lossy part of the network at the link level with Forward Error Correction (FEC) codes [26]. The idea behind FEC is to send, in addition to the original data, some redundant information so that a packet, corrupted while crossing a wireless link, can be reconstructed at the output of the link without requiring any retransmission. One can see FEC as sending, together with original packets, copies of them so that the copy can be used when the original packet is lost, of course if the copy itself is not lost. The use of FEC should improve the quality of the noisy link while consuming some extra bandwidth. The

other drawback of FEC is that it requires some processing time for coding and decoding the redundant information. However, the advantages of FEC are numerous and make it the most interesting solution despite its cost [26, 34]. The first advantage of link-level FEC is that it does not require any change of the TCP code. Other solutions as Explicit Loss Notification [26] require such a change. The second advantage of FEC is that it does not require any monitoring of TCP packets inside the network. The monitoring of TCP packets at the entry of the lossy part of the network is used by the other solutions either to split the TCP connection [24] or to retransmit corrupted packets on behalf of the source [26]. Hence, FEC respects the principle of layering and the end-to-end semantics of TCP since it can be entirely implemented below the IP layer without the need to look at TCP and IP headers. The third advantage of FEC is that the correction of packets is done on runtime which eliminates any interference with TCP error recovery mechanisms. Note that a solution that retransmits packets on behalf of the source at the entry of the lossy part of the network (e.g., Snoop protocol [26]) causes fluctuations of end-to-end delay and possible expirations of TCP retransmission timer. These fluctuations are much more important on paths where the propagation delay of the wireless link is large (e.g., satellite links). Note also that the retransmission of packets within the network might cause the generation of duplicate ACKs when the link layer continues to deliver packets to the receiver while the corrupted packet is being retransmitted over the lossy link. These problems of retransmission-based solutions are solved with FEC since packets are delivered to upper layers with the same order with which they are transmitted over the wireless link. Of course, we may obtain gaps at the receiver which correspond to packets that could not be reconstructed. These advantages of FEC make from it a recommended solution for the improvement of the quality of bad transmission media especially those of long propagation delay [7]. For example, Convolutional coding, Viterbi decoding, together with interleaving techniques and Reed-Solomon encoding, are widely used to render satellite links and wireless links as clean as terrestrial ones.

In this chapter, we address the problem of bandwidth tradeoff between link-level FEC and TCP congestion control. While consuming some extra bandwidth, FEC improves the throughput of TCP by shielding it from non-congestion losses. However, much FEC may steal some of the bandwidth that was used by TCP. The question that we ask is, given a certain noisy link with certain characteristics (bandwidth, error rate, burstiness of errors), how to choose the amount of FEC so that to get the maximum gain in TCP performance. Our aim is to understand the relation between the bandwidth consumed by FEC and that gained by TCP. We propose for this purpose two models: one for non-congestion losses and one for FEC. We then use some previous results from end-to-end modeling of TCP and some mathematical tools to approximate the throughput of a TCP connection as a function of the amount of redundancy we add on the wireless link and as a function of the way with which non-congestion losses occur (i.e., rate and burstiness of losses). This gives insights on how to tune the FEC mechanism so that TCP transfers achieve the best performance. The different results of our study are validated with `ns`

simulations.

9.1 The model

Consider a TCP connection that crosses a network including a noisy wireless link of rate μ (packets/s). We suppose that the quality of the noisy link is improved by a certain amount of FEC. In the next two sections, we define our model for the loss process over the wireless link as well as our model for the FEC added to improve its quality.

9.1.1 The model for non-congestion losses

It is well known that transmission errors over wireless links tend to appear in bursts [49, 51, 55, 83]. The model often used in the literature to represent correlated losses on wireless links is the one introduced by Gilbert [49, 51, 66, 83]. We already referred to such model when we presented our Markovian model for the calculation of TCP throughput (Chapter 4). The Gilbert model is a simple ON/OFF model. The lossy link is supposed to be in one of two states: Good and Bad. A packet is lost if it leaves the link while it is in the Bad state otherwise it is supposed to be correctly received. A discrete time Markov chain with two states (Good and Bad) is used to model the dynamics of the wireless link (Figure 9.1). This gives a Markovian model similar to the one we used in Chapter 4 to study the fluctuations of the loss rate on the path of a TCP connection. The difference is that in the Gilbert case all packets are lost and only lost in the Bad state of the link, whereas in our end-to-end model packets can be lost in both states with a higher probability in the Bad state than in the Good state.

We focus on the loss process of link-level packets also called *transmission units*. We suppose that a TCP packet is transmitted over the wireless link using multiple small transmission units [49, 51]. A transmission unit can be a bit, a byte, an ATM cell, or any other kind of link-level blocks used for the transmission of TCP/IP packets. The state of the wireless link (Good or Bad) is observed upon the arrival of transmission units at its output. We suppose that units cross continuously the link. If no real units exist, *fictive units* are inserted. In other words, transitions of the Markov chain associated to the wireless link happen at deterministic moments. The time between two transitions is equal to the transmission time of a unit on the wireless link.

For clarity, we use the same notation for the transition probabilities of the Markov chain as those used in Chapter 4. Let $\bar{\gamma} = 1 - \gamma$ denote the probability that the noisy link passes from Good state to Bad state when a transmission unit arrives at its output. Let $\beta = 1 - \bar{\beta}$ denote the probability that it stays in the Bad state. According to what transmission units mean, β and γ can be one of the quantities used to measure error rates in real networks (Bit Error Rate, Cell Loss Ratio, etc.). β represents how much the loss process of transmission units is bursty. A β close to zero means that losses are isolated and a β close to 1 means that long bursts are dominant. Now, a β equal to $\bar{\gamma}$ brings us to the case of a memoryless loss process where units

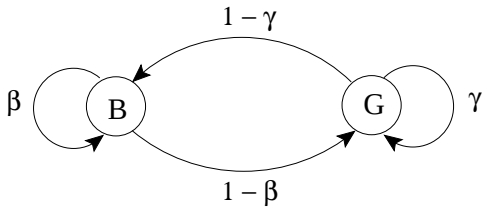


Figure 9.1: The Gilbert loss model

are lost independently of each other (Bernoulli loss process).

We give the expressions of the stationary probabilities that the link is in the Bad and Good states. We suppose that $\beta, \gamma \in (0,1)$ so that the Markov chain associated to the lossy link is ergodic and that the stationary probabilities exist and are unique. These probabilities, denoted π_B and π_G respectively, are used in the following to study what happens to a unit regardless of what has happened to the preceding one. We have,

$$\pi_B = \frac{\bar{\gamma}}{\bar{\beta}+\bar{\gamma}}, \qquad \pi_G = \frac{\bar{\beta}}{\bar{\beta}+\bar{\gamma}}.$$

It is interesting to calculate the average lengths of Good and Bad periods in terms of transmission units. We denote these lengths by L_B and L_G respectively. A simple calculation shows that,

$$L_B = \frac{1}{\bar{\beta}}, \qquad L_G = \frac{1}{\bar{\gamma}}. \tag{9.1}$$

The expressions of L_B and L_G permit us to calculate the average loss rate as a function of β and γ. Denote this rate by l. Intuitively, it is equal to the probability that the link is in the Bad state regardless of what has happened to the previous unit. We have,

$$l = \frac{L_B}{L_B + L_G} = \frac{\bar{\gamma}}{\bar{\beta}+\bar{\gamma}} = \pi_B. \tag{9.2}$$

We use the expression of l to change the burstiness of losses while maintaining the same loss rate. It is clear that the increase in β stretches the duration of the Bad state which increases the burstiness of losses. For a certain loss rate l, we vary β from 0 to 1 in order to increase the burstiness. Then for each value of β, we use the expression of l to calculate the value of γ that gives the same loss rate.

9.1.2 The FEC model

The most common code used for error correction is the *block* code [114, 119]. Suppose that data are transmitted in units as in our model for the lossy link. Block code FEC consists in grouping the data units in blocks of K units. Then, a codec adds to every block a group of R redundant units calculated from the K original units. The result is the transmission of blocks of total size $N = K + R$ units (Figure 9.2). At the receiver, the original K units of a block are reconstructed

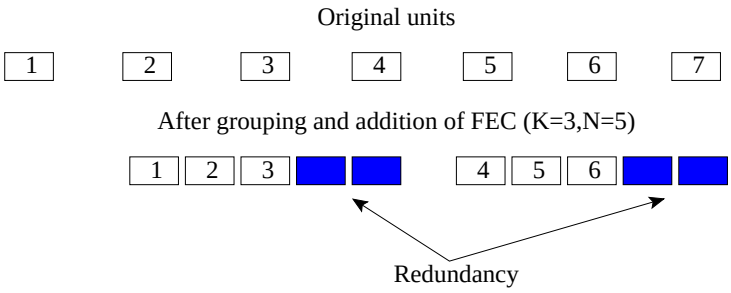


Figure 9.2: Operation of a FEC codec

if at least K of the total N units it carries are correctly received. This should improve the quality of the link since a block can now resist to R losses without being discarded.

Block FEC can be implemented in the physical layer to recover from the loss of bits (e.g., Reed-Solomon codes). This implementation, together with interleaving, is used in satellite links to correct the bursts of bit errors that result from the utilization of Convolutional coding/Viterbi decoding. Block FEC can be also implemented at a higher level to recover from the loss of frames or packets. This latter implementation of FEC is known as the recovery from packet errors or simply from erasures [114, 119]. An example of erasure block FEC is the one proposed in [103] for the recovery from lost ATM cells. The difference between the two implementations of FEC is that in the first case a frame contains the redundancy and the original data. However, in the second case, the redundancy is included in other frames. Adding redundancy to other frames helps the higher layers to recover from losses not only caused by corruption but also by other phenomena such as congestion of switches or the MAC (Medium Access Control) layer.

Consider a block FEC code implemented in the same layer as transmission units. We ignore any FEC code that may exist below this layer. The input to our study is the loss process seen by transmission units which we assumed to follow the Gilbert model (Figure 9.1). The FEC layer at the entry of the lossy link (codec) groups the transmission units in blocks of size K, then it adds R redundant units to each block. The total number of units in a block becomes equal to $N = K + R$. At the output of the lossy link, another FEC layer (decodec) takes the N units in each block, eliminates the redundancy, and delivers the original block to the upper layer. A block is delivered if at least K of its N units are correctly received, otherwise it is supposed to be discarded. In what follows, we study how much such FEC scheme impacts the performance of TCP.

9.2 The approximation of TCP throughput

Consider the throughput of the connection as the performance measure that indicates how well TCP behaves over the wireless link. In the first part of this thesis, we saw different models for the calculation of TCP throughput [12, 84, 92, 105]. These models assume idealized TCP

sources and find the expression of the throughput as a function of the probability that a TCP packet is lost inside the network. Denote this probability by p. In case of bursty losses, p represents the inverse of the average number of TCP packets correctly received between two bursts of losses [105]. In other words, p represents the probability that a packet is the first loss in a burst of losses. This is because the new versions of TCP (e.g., New-Reno [64], SACK [56, 91]) are designed in a way to divide the window once by two for a burst of packet losses. The different models show that the throughput of TCP is inversely proportional to the square root of p as well as to the average round-trip time. The difference between the different derived expressions for TCP throughput is in the number of factors the models consider. We refer to Chapter 3 for a description of the different factors a model for TCP has to consider. The simplest expression for TCP throughput [92], called the *square root formula*, only considers the linear-increase multiplicative-decrease part of TCP congestion control and makes the assumption that the moments at which the window of TCP is reduced are equally separated. This simple expression gives good performance when p is small [105] and when the times between the moments at which the window of TCP is reduced (i.e., congestion moments) do not vary very much [12]. The other expressions of the throughput as the ones we found in the first part of this thesis, consider more factors as the timeout phenomenon, the receiver window, the possibility of other distributions of losses, etc.

Without loss of generality, we consider the simple square root formula for TCP throughput. One can simply use other more sophisticated expressions of the throughput. For example, one can use the formula in [105] or a particular case of our general expression in Chapter 5. Suppose that the receiver acknowledges every data packet. Let T denote the average number of packets correctly received between losses ($T = 1/p$). In case of bursty losses, T denotes the average number of packets between bursts of losses. Let RTT denote the average round-trip time seen by the connection. Thus, we can write the throughput of TCP in terms of packets/s as [92]

$$\bar{X} = \frac{1}{RTT}\sqrt{\frac{3}{2}T} = \frac{1}{RTT}\sqrt{\frac{3}{2P}}.$$

Suppose that the wireless link is the bottleneck on the path of the connection. We make this assumption since our objective is to optimize the amount of FEC so that a TCP connection becomes able to fully utilize the available bandwidth on the wireless link. In general, this assumption holds given the scarcity of resources on air interfaces. Thus, in the absence of FEC, the throughput of TCP is upper bounded by μ and we write it as follows

$$\bar{X} = \min\left(\frac{1}{RTT}\sqrt{\frac{3}{2}T}, \mu\right).$$

Our idea is to find the expression of TCP throughput as a function of the parameters of the loss process of transmission units (β, γ) and the parameters of the FEC scheme (N, K). With such expression, we can optimize the amount of FEC so that to get the best throughput and

this is for a given process of errors on the wireless link. We already have the expression of the throughput as a function of what happens at the packet level (i.e., as a function of p). What we still need to do is to relate the loss process of transmission units to the loss process of TCP packets. But, TCP packets can be lost due to congestion in other parts of the network not only on the wireless interface. The loss process of TCP packets is the sum of multiple processes, one for each part of the network. To simplify the analysis, we consider the best case when packets are only lost on the wireless interface and this is whenever the wireless interface is not fully utilize. When the wireless interface becomes fully utilized, congestion losses may appear in other parts of the network but this will not impact the throughput of TCP since it will remain equal to the available bandwidth on this interface. Note here that adding a certain amount of FEC when packets are lost in other parts of the network gives less gain in TCP performance than when packets are only lost on the wireless link. This is simply because the relative increase in the total packet loss probability p caused by the addition of FEC is less important in the first case. For example, if packets are most probability lost due to congestion, adding FEC to the wireless link will not improve the throughput since it is only determined by congestion losses.

Using our above assumptions, we calculate T as a function of the different parameters of the Gilbert and FEC models. This gives the expression of the throughput of TCP. In the case transmission units are lost independently of each other ($\beta = 1 - \gamma$), the calculation of T is straightforward; $p = 1/T$ is no other than the probability that a TCP packet is lost while crossing the wireless link. This case is studied in the next section. The difficulty exists when transmission units are lost in bursts ($\beta > 1-\gamma$). The correlation of losses at the unit level causes the correlation of losses at the TCP packet level. T must be then calculated as the average number of TCP packets correctly transmitted between bursts of packet losses. But, because of redundancy and because of the notion of blocks and units, the calculation of T in the bursty case is quite difficult. Some further assumptions must be made at the unit level and at the packet level in order to ease the analysis. This is done in Section 9.4. Note here that our aim from the study of the correlation case is to evaluate the impact of burstiness of transmission errors on the efficiency of a given FEC scheme, and hence on the throughput of TCP. We are not interested in the study of the effect of the correlation of packet losses at the TCP level on end-to-end performance. One can look at the simulations in [56] for the understanding of such effect. Normally, a correlation of losses at the packet level should not deteriorate the performance since new versions of TCP are designed in a way to reduce the window once by two for a burst of packet losses in the same round-trip time.

Now, even though it increases T, the addition of FEC consumes some bandwidth and decreases the maximum throughput TCP can achieve. Instead of μ, we get $K\mu/N$ as a maximum TCP throughput. If we denote by S the size of a TCP packet in terms of transmission units, the

throughput of TCP in presence of FEC and in terms of units/s can be written as,

$$\bar{X}(N,K) = \min\left(\frac{S}{RTT}\sqrt{\frac{3}{2}T(N,K)}, \frac{K}{N}\mu\right). \tag{9.3}$$

Along the chapter, TCP throughput and rates will be expressed in terms of transmission units per second. For given Gilbert model parameters, we will focus on the calculation of $T(N,K)$, and hence the throughput $\bar{X}(N,K)$, as a function of the amount of FEC. We start first by the non-correlation case. Second, we consider the case when transmission errors are correlated.

9.3 The case of non-correlated losses

In this section, we suppose that transmission units are lost independently of each other with probability $\beta = \bar{\gamma}$. Thus, TCP packets are also lost independently of each other but with a probability $p(N,K)$ which is a function of the FEC scheme (N,K) we are using on the wireless link. The throughput of the TCP connection can be approximated by substituting T in formula (9.3) by $1/p(N,K)$. First, we state our analysis of TCP performance as well as some numerical results. Next, simulation results are presented.

9.3.1 The analysis

Suppose that TCP packets are of the size of one link-level block ($S = K$ units). Given a certain block size (K) and a certain FEC rate (K/N), the choice of the size of the TCP packet in terms of blocks is another problem that we will not address in this chapter. It is a problem of TCP algorithms rather than the amount of FEC we are using on the link. However, we noticed that with the values of K we use in this chapter, larger packets give approximately the same performance as single-block packets.

A packet is lost when more than R of its units are lost due to transmission errors. This happens with a probability,

$$p(N,K) = \sum_{i=0}^{K-1} \binom{N}{i} (1-\beta)^i \beta^{N-i}.$$

$\binom{N}{i}$ is the binomial coefficient equal to $N!/(i!(N-i)!)$. The throughput of TCP is obtained by plugging this value of $p(N,K)$ in Equation (9.3).

It is clear from the expression of $p(N,K)$ that the addition of FEC (i.e., N>K) at the link level reduces the loss probability of TCP packets and hence increases the throughput. This happens whenever the first term of the minimum function in (9.3) is smaller than the second term. The improvement of the throughput continues until the two terms of the minimum function become equal. At this point, the quantity of FEC added to the wireless link is sufficient to eliminate the negative effect of non-congestion losses on TCP. We say here that the FEC has *cleaned* the noisy link from TCP point of view. Any increase in N beyond this point results in a throughput

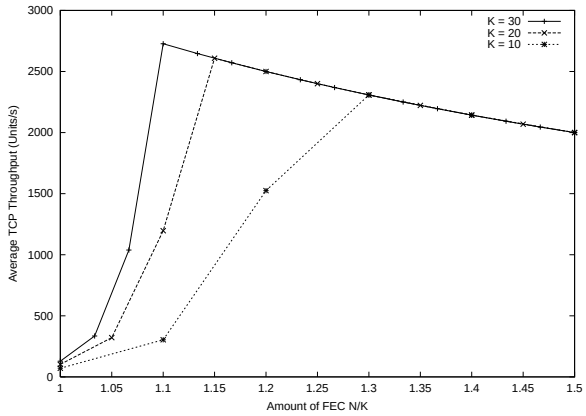


Figure 9.3: Model: $\bar{X}$ vs. N/K and K

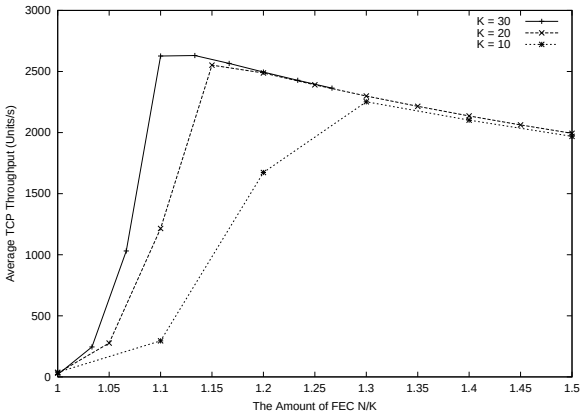


Figure 9.4: Simulation: $\bar{X}$ vs. N/K and K

deterioration since the throughput is equal to the second term of the minimum function in (9.3) which in turn decreases with N. There will be more FEC than what is needed to clean the link. Thus, the appropriate amount of FEC is the one given by the equality of the two terms of the minimum function. Given a wireless link of bandwidth μ, transmission blocks of size K, and a unit loss probability β, the optimal amount of FEC from TCP point of view is given by the solution of the following equation,

$$\frac{N}{RTT}\sqrt{\frac{3}{2p(N,K)}} = \mu. \tag{9.4}$$

Note that such amount of FEC is only optimal for TCP connections. Non-TCP transfers as real time audio and video flows may require another amount of FEC function of the loss rate they tolerate.

9.3.2 Analytical results

We show in Figure 9.3 how the throughput of TCP varies as a function of the ratio N/K (the quantity of FEC) for different values of K (10, 20, and 30 units). RTT is taken equal to 560 ms and the wireless link bandwidth to 3000 units/s. We can consider this scenario as the case of a mobile user downloading data from the Internet using a satellite link. This particular value of μ is approximately equal to the maximum ATM cell rate on a T1 link (1.5 Mbps). The unit loss probability β is set to 0.01.

It is clear that the performance of TCP improves considerably when FEC is added and that this improvement continues until the optimum point given by Equation (9.4) is reached. Beyond this point, any increase in FEC deteriorates the throughput as we explained. Also, we notice that for a certain quantity of FEC, an increase in K improves the performance of TCP. An increase in the block size results in a larger R, thus in a better capacity to correct multiple errors per packet. At large blocks, FEC can correct the same errors corrected when blocks are

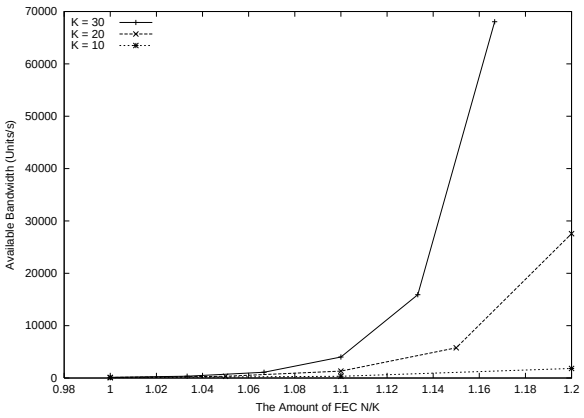


Figure 9.5: Model: Optimal FEC vs. μ

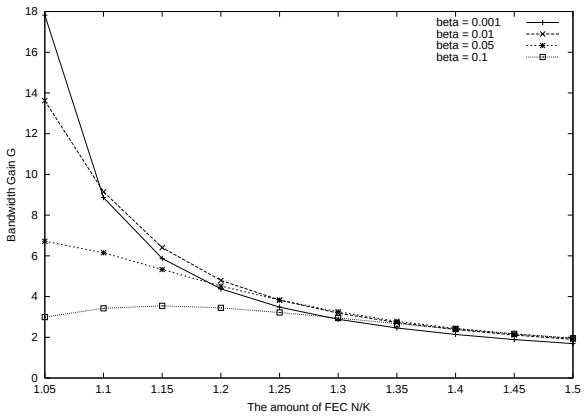


Figure 9.6: Model: G vs. N/K and β

divided into small ones but it also has the capacity to correct these errors when they are grouped together. Another reason for the improvement of the performance is that an increase in K at a constant FEC rate results in larger TCP packets, hence in a faster growth of the congestion window. Recall that TCP window is increased in terms of packets rather than bytes. Thus, the source returns faster to its rate prior to the detection of a packet loss. But, increasing the block size has some problems. It yields longer end-to-end delays (more time is needed to fill a block, see Figure 9.2), more processing time to code and decode the redundancy (redundant units are computed as a function of all the original units in the block), and larger memory requirements (the units of a block must be stored at the input before being coded and transmitted on the wireless link and at the output before being decoded and handed to the upper layers).

In Figure 9.5, we plot the left-hand side of Equation (9.4) as a function of N/K for the same three values of K. These curves provide us with the optimal amount of FEC for a given μ, β, and K. The optimal amount of FEC is obtained by the abscissa of the intersection point of the curve corresponding to K and the horizontal line corresponding to μ. We see well that any increase in K reduces considerably the amount of FEC needed to clean the wireless link from TCP point of view. Again, this is because the increase in K results in a faster growth of the congestion window and in a better resilience against grouped errors. Given a certain μ, a compromise between K and FEC rate must be done. First, we have to choose the largest possible K, then we choose the appropriate amount of FEC.

For $\mu = 3000$ units/s and $K = 20$, we show in Figure 9.7 how the throughput of TCP varies as a function of the transmission unit loss probability β and this is for different values of N. It is clear that adding just one redundant unit to every FEC block results in a considerable gain in performance especially at small β. Adding more redundancy at small β deteriorates slightly the performance since the link is already clean and the additional redundancy steals some of the bandwidth used by TCP. This is not the case at high β where much redundancy needs to be

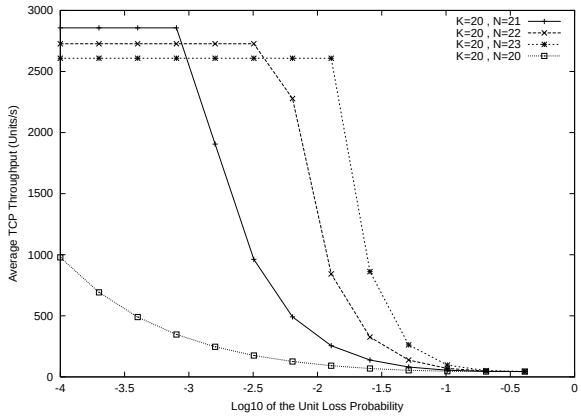


Figure 9.7: Model: $\bar{X}$ vs. β and N/K

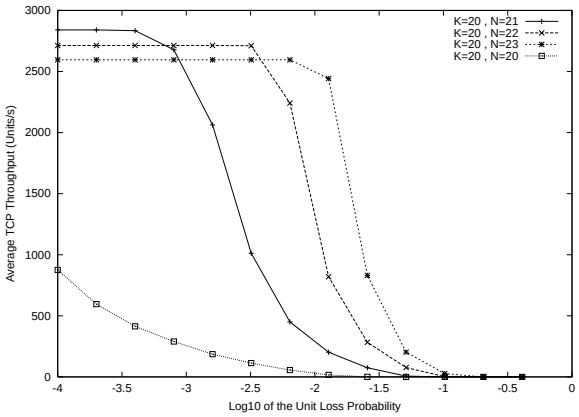


Figure 9.8: Simulation: $\bar{X}$ vs. β and N/K

used in order to get good performance. We see well how the situation changes in the middle of the β axis and how a large N starts to give better performance. Note that even though an excess of FEC reduces the performance of TCP when losses are rare, the reduction is negligible compared to the gain in performance we obtain when losses become frequent. When the link is heavily lossy ($\log(\beta) > -1.7$), the three amounts of FEC plotted in the figure become insufficient to clean the wireless link and all the curves converge asymptotically to the same point. This asymptotic behavior is due to the fact that the packet loss probability $p(N, K)$ tends to 1 when β tends to one whatever are the values of N and K.

9.3.3 Simulation results

Using `ns` [102], we simulate a simple scenario where a TCP source is connected to an IP router via a high speed wire line and where the router is connected to the TCP receiver via a lossy wireless link. The Reno version of TCP [56] is used. This version tries to avoid slow start when recovering from losses and hence must give close results to the expression of the throughput we used. Recall that this expression is derived under the assumption that the source always stays in the congestion avoidance phase [92]. The TCP source is fed by an FTP application with an infinite amount of data to send. We parameterize the TCP receiver so that it acknowledges all data packets and advertises an infinite window. We then add our FEC model to the simulator. A TCP packet is fragmented at the entry of the wireless link into K units. The FEC layer adds then R redundant units to each packet (block). At the output of the wireless link, the FEC layer reconstructs the original packet if at least K of its N units are correctly received otherwise it rejects it. The transmission units on the lossy link are supposed to be ATM cells of size 53 bytes (a payload of 48 bytes and a header of 5 bytes). We choose the bandwidth of the lossy link in a way to get a transmission rate μ equal to 3000 cells/s. This corresponds to a 1.5 Mbps T1 link. RTT is taken equal to 560 ms and the buffer size in the middle router is set to 100 packets. This

buffer size is larger than the bandwidth-delay product (μRTT) which guarantees that no losses occur in the middle router before the full utilization of the available bandwidth on the wireless interface. This also guarantees that after the division of the window by two due to a congestion event, the wireless link remains fully utilized. This satisfies what we wrote in (9.3).

Figures 9.4 and 9.8 show the variation of the simulated throughput as a function of the amount of FEC (N/K) and the unit loss probability β. In the first figure, β is set to 0.01. We notice clearly the good match between these results and the analytical ones. The small difference is due to the fact that the expression of the throughput we used in our analysis does not consider the possibility of a timeout when multiple packet losses appear in the same round-trip time. One can account for timeouts by using the correction we introduced in Chapter 3 (Equation (3.1)). Moreover, in our analysis we considered that RTT is always constant which does not hold when the throughput of TCP approaches the available bandwidth. To get a throughput (i.e., an average transmission rate) close to the available bandwidth, the TCP source needs to transmit during some time at a rate higher than the available bandwidth. This is due to the saw-tooth nature of TCP congestion control. Transmitting at higher than the available bandwidth means some queueing time which disappears when the rate of TCP is reduced to less than the available bandwidth. One can use a fixed-point approach to get a more precise expression of the throughput when we are close to the full utilization of the available bandwidth. Due to the increase in the round-trip time, we must expect a slightly lower throughput in this region than that given by (9.3). This can be seen in both figures when we approach the points of optimal FEC. The fixed-point approach tells us that RTT is equal to the propagation delay when the throughput is less than 3/4 the available bandwidth on the wireless interface.

9.3.4 The tradeoff between TCP throughput and FEC cost

We compare in this section the bandwidth gained by TCP to that consumed by FEC. Let G be the ratio of these two bandwidths,

$$G = \frac{\bar{X}(N,K) - \bar{X}(K,K)}{\bar{X}(N,K) \times \frac{N-K}{K}} = \left(1 - \frac{\bar{X}(K,K)}{\bar{X}(N,K)}\right) \times \left(\frac{K}{N-K}\right). \tag{9.5}$$

This quantity indicates how much beneficial is the addition of FEC. It can be considered as a measure of the overall performance of the system TCP-FEC. We want to improve TCP performance without paying for FEC more than we gain in TCP performance. A value close to one of this gain means that we pay for FEC as much as we gain in TCP throughput. A negative value can mean that the link was clean for TCP so that the addition of FEC has reduced the performance instead of improving it. It can also mean that FEC is added in large quantities so that it steals some of the bandwidth used by TCP.

In Figure 9.6, we plot G as a function of the amount of FEC for different unit loss probabilities. Again, we take $\mu = 3000$ units/s and $K = 20$. This figure shows that the gain in overall performance is important when the loss probability and the amount of FEC are small. Moreover,

with small amount of FEC, the gain decreases considerably when the loss rate ($l = \beta$) increases. Now, when the amount of FEC increases, the curves converge approximately to the same point with a slightly better gain this time for high loss probabilities.

For small β, little FEC is sufficient to clean a link which improves considerably the performance of TCP. On the other hand, this little FEC is not able to clean a link with a high β. The result is a small gain in TCP performance, hence a small G. This explains what we see in the left-hand part of Figure 9.6.

Consider now the right-hand part of the figure. A large amount of FEC is able to clean links with a wide range of β. TCP then obtains the same throughput $\bar{X}(N,K)$ — equal to $K\mu/N$ units/s — for all the values of β we consider in the figure. But, the gain we defined in Equation (9.5) is not only a function of the throughput after the addition of FEC but also a function of the throughput before this addition. Given that the initial throughput — denoted by $\bar{X}(K,K)$ in the equation defining G — decreases when β increases, the gain in overall performance is more important at high β than at small β. The point at which two curves in Figure 9.6 meet gives us the required amount of FEC to clean their corresponding links.

9.3.5 Number of connections and the gain in performance

We notice in Figure 9.6 that using a small amount of FEC gives the best gain in overall performance. A small amount of FEC helps TCP to improve considerably its throughput but it does not help it to use all the available bandwidth. Any additional FEC improves further the quality of the link which improves further the throughput of TCP but the revenues are not as important as for the first units of redundancy. Thus, in order to maintain a high gain, one can use a small amount of FEC and share the available bandwidth between multiple TCP connections. The result will be a better utilization of the link bandwidth while using a small amount of FEC. But in practice, one cannot guarantee that there are always enough connections to use the available bandwidth. A TCP connection must be able to use all the bandwidth when it operates alone in the network. For this reason, FEC has to be added in large amounts so that to make the lossy link clean from the point of view of a single TCP connection even if the achieved gain is not so important.

To clarify this point, we study the gain in the overall performance when many TCP connections share the lossy link. Suppose that C connections run simultaneously between the source and the destination. Because they see the same RTT and they are subject to the same loss process, we can suppose that the C connections achieve the same throughput [84]. Let $p_i(N,K)$ be the probability that a packet from an individual connection is lost. Using (9.3), the total TCP throughput can be written as

$$\bar{X}(N,K) = \min\left(\frac{CS}{RTT}\sqrt{\frac{3}{2p_i(N,K)}}, \frac{K}{N}\mu\right). \tag{9.6}$$

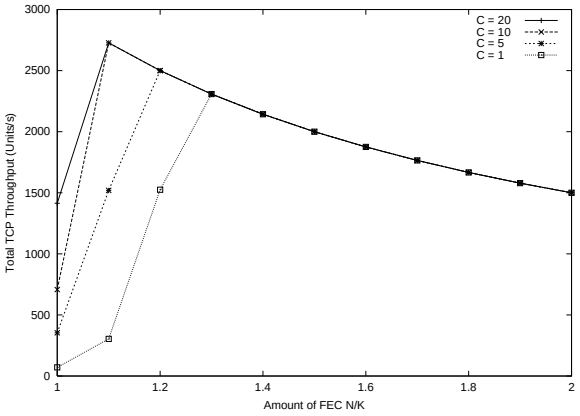


Figure 9.9: Model: $\bar{X}$ vs. N/K and C

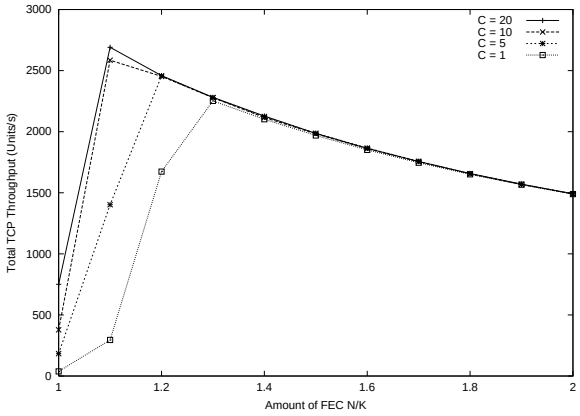


Figure 9.10: Simulation: $\bar{X}$ vs. N/K and C

Because the loss process is not correlated, $p_i(N,K)$ is simply equal to $p(N,K)$. Thus, the difference in the case of many connections than in the case of a single one is a multiplicative factor C in the first term of the minimum function in Equation (9.3). The second term of this function remains unchanged. A factor C means a faster improvement of the throughput when the amount of FEC increases as illustrated in Figure 9.9. This figure corresponds to $\beta = 0.01$ and $K = 10$. At large C, the link can be cleaned by a smaller amount of FEC than at small C. When the amount of FEC added to a link is not enough to clean it, one can open multiple TCP connections to use more bandwidth and get more gain in overall performance instead of adding more FEC to improve the link quality. Note here that adding more FEC to clean a link reduces the maximum limit of TCP throughput ($\mu K/N$). However, using multiple TCP connections does not change this maximum limit. As we see in Figure 9.9, a total throughput of 2700 units/s cannot be obtained when few connections are sharing the wireless link even if enough FEC is added to clean it. However, this total throughput can be obtained when increasing the number of connections and reducing the amount of FEC.

In Figure 9.10, we show the simulation results that correspond to Figure 9.9. The match is clear for large amounts of FEC. However for small amounts, the analysis gives larger values. As we already explained, this discrepancy is due to the fact that the expression of the throughput we used in our analysis does not consider the timeout phenomenon. At small amount of FEC, losses are frequent which results in small windows, multiple losses per window, and an important probability of timeout [105]. This leads to poorer throughput than the one given by the analysis. The difference in the results is exacerbated by the factor C which explains the large mismatch in the case of 20 connections.

9.4 The case of correlated losses

In this section, we study the effect of burstiness of transmission errors on the efficiency of a FEC scheme. It is clear that when unit losses tend to appear in bursts, more redundant information is needed to clean the wireless link. Packets are hurt by burst of losses and hence they require a large number of redundant units per packet (R) to be corrected. But, for the same average loss rate (l), the correlation of losses reduces the probability that the link passes from the Good state to the Bad state ($\bar{\gamma}$ decreases when β increases, see (9.2)). This reduces the probability that a packet is hurt by a burst of losses. TCP throughput may then improve and the amount of FEC may be reduced. Thus, an analysis is needed to understand these opposite effects of burstiness.

9.4.1 Performance analysis

Burstiness at the unit level results in burstiness at the TCP level. In the presence of TCP versions that divide their windows once by two in response to a burst of packet losses (e.g., SACK [56]), T in Equation (9.3) represents the average number of TCP packets correctly received between two bursts of packet losses. Let us calculate T, and hence the throughput, as a function of the amount of FEC, the packet size K, the average loss rate l, the burstiness of losses β, and the total bandwidth μ.

Calculation of the average number of packets between bursts

Let t be the number of TCP packets correctly received between two separate bursts of losses at the TCP level. The minimum value of t is therefore one packet and its expectation is equal to T. Let Y_n be the state of packet n. Let 0 be the number of the first good packet between the two bursts. Y_n takes one of two values: G (Good) and B (Bad). We have $Y_0 = G$. The expectation $T = \mathrm{E}[t]$ can be written as,

$$T = \sum_{n=0}^{\infty} \mathrm{P}\{t > n | Y_0 = G\} = 1 + \sum_{n=1}^{\infty} \mathrm{P}\{t > n | Y_0 = G\}.$$

The computation of T is quite complicated since the spacing between the TCP packets varies with the window size. Another complication is that $\{Y_n\}$ does not form a Markov chain. Indeed, if we know for example that a packet n is of type B, then the probability that packet $n+1$ is of type G also depends on the type of packet $n-1$. If packet $n-1$ were G rather than B, then the *last units of packet* n are more likely to be those that caused its loss. Hence, the probability that packet $n+1$ is B is larger in this case.

This motivates us to introduce another random variable which will make the system more "Markovian" and will permit us to write recurrent equations in order to solve the problem for T. We use for this purpose the state of the *last transmission unit* received, or fictively received, before a TCP packet. The knowledge of the state of this unit, denoted by Y_n^{-1} and which takes

the values B and G, fully determines the distribution of the state Y_n of the following TCP packet. Using the state of this particular unit, we write T as

$$T = 1 + u\mathrm{P}\left\{Y_1^{-1} = G|Y_0 = G\right\} + v\mathrm{P}\left\{Y_1^{-1} = B|Y_0 = G\right\},$$

where

$$u = \sum_{n=1}^{\infty} \mathrm{P}\left\{t > n|Y_0 = G, Y_1^{-1} = G\right\},$$

$$v = \sum_{n=1}^{\infty} \mathrm{P}\left\{t > n|Y_0 = G, Y_1^{-1} = B\right\}.$$

We shall make throughout the following assumption,

Assumption 9.1

$$\mathrm{P}\left\{Y_1^{-1} = G|Y_0 = G\right\} \approx \pi_{_G},$$

$$\mathrm{P}\left\{Y_1^{-1} = B|Y_0 = G\right\} \approx \pi_{_B}.$$

Assumption 9.1 holds when the time to reach steady state for the Markov chain in the Gilbert model is shorter than the time between the beginning of two consecutive TCP packets (either because the TCP packets are sufficiently long, or because the TCP packets are sufficiently spaced). Assumption 9.1 also holds when $\pi_{_B}$ and the loss probability of a whole TCP packet are small. Indeed, we can write,

$$\begin{aligned}\pi_{_G} &= \mathrm{P}\left\{Y_1^{-1} = G|Y_0 = G\right\}\mathrm{P}\left\{Y_0 = G\right\} + \mathrm{P}\left\{Y_1^{-1} = G|Y_0 = B\right\}\mathrm{P}\left\{Y_0 = B\right\} \\ &\approx \mathrm{P}\left\{Y_1^{-1} = G|Y_0 = G\right\} \cdot 1 + \mathrm{P}\left\{Y_1^{-1} = G|Y_0 = B\right\} \cdot 0\end{aligned}$$

In view of Assumption 9.1, the probability that the unit preceding a packet is lost can be considered as independent of the state of the previous packet. It follows that,

$$T = 1 + u\pi_{_G} + v\pi_{_B},$$

$$u = (1 - \mathrm{P}\left\{Y_1 = B|Y_1^{-1} = G\right\})(1 + u\pi_{_G} + v\pi_{_B}),$$

$$v = (1 - \mathrm{P}\left\{Y_1 = B|Y_1^{-1} = B\right\})(1 + u\pi_{_G} + v\pi_{_B}),$$

which gives us,

$$\frac{1}{T} = \pi_{_G}\mathrm{P}\left\{Y_1 = B|Y_1^{-1} = G\right\} + \pi_{_B}\mathrm{P}\left\{Y_1 = B|Y_1^{-1} = B\right\}.$$

The calculation of T, and therefore of the throughput, is thus simplified to the calculation of the probability that a packet is lost given the state of the unit just preceding it. These are the two probabilities $\mathrm{P}\left\{Y_1 = B|Y_1^{-1} = G\right\}$ and $\mathrm{P}\left\{Y_1 = B|Y_1^{-1} = B\right\}$ that figure in the above expression of $1/T$. But again, it is difficult to find explicit expressions for these two probabilities. A TCP

packet can be lost by a single long burst of unit losses as well as by multiple separate small bursts.

To further facilitate the analysis, we assume that bursts of losses at the unit level are quite separated so that two bursts rarely appear within the same packet. This is formalized in the following assumption,

Assumption 9.2 *The following holds,*

$$\bar{\beta} \cdot l \cdot N << 1.$$

Let us prove the condition we wrote for our second assumption to hold. A TCP packet is supposed to be only lost if it is hurt by a burst larger than R. Therefore, we don't consider the probability that multiple small and separate bursts at the unit level contribute to the loss of a TCP packet. This is possible when the sum of the average lengths of the Good state (L_G) and the Bad state (L_B) is much larger than the packet length N. Using (9.1) and (9.2), we get the condition in Assumption 9.2. If Assumption 9.2 is not satisfied, many bursts may appear within the same packet leading to a higher loss probability of a TCP packet than the one given by our analysis. In this case, we expect our analysis to result in an overestimation of the real throughput.

Consider first the case $Y_1^{-1} = B$. In view of Assumption 9.2, packet 1 is lost if its first $R+1$ units are also lost. Thus,

$$\mathrm{P}\left\{Y_1 = B | Y_1^{-1} = B\right\} = \beta^{R+1}.$$

For the case $Y_1^{-1} = G$, packet 1 is lost if a burst of losses of length at least equal to $R+1$ units appears in its middle. We get,

$$\mathrm{P}\left\{Y_1 = B | Y_1^{-1} = G\right\} = \beta^R \bar{\gamma} \left(1 + \gamma + \cdots + \gamma^{N-R-1}\right) \simeq K \beta^R \bar{\gamma}.$$

We used for this result the approximation $(1 - \gamma^{N-R}) \simeq K(1-\gamma)$.

Given a certain average loss rate l and a certain correlation of losses expressed by β, we can find γ using Equation (9.2). Note that the ratio of $\bar{\beta}$ and $\bar{\gamma}$ needs to be maintained constant for the loss rate to remain the same. T can then be approximated for any FEC scheme (N,K) by,

$$\frac{1}{T} = \beta^{N-K} l \left(\bar{\beta} K + \beta\right). \tag{9.7}$$

The throughput can be approximated by plugging this value of T in Equation (9.3).

9.4.2 Analytical results

Using (9.3) and (9.7), we plot in Figure 9.11 the throughput of TCP as a function of burstiness and this is for different FEC rates. The burstiness is varied by varying β which is called the Conditional Loss Probability in the figure. K is set to 20 and the loss rate l to 0.01. γ is

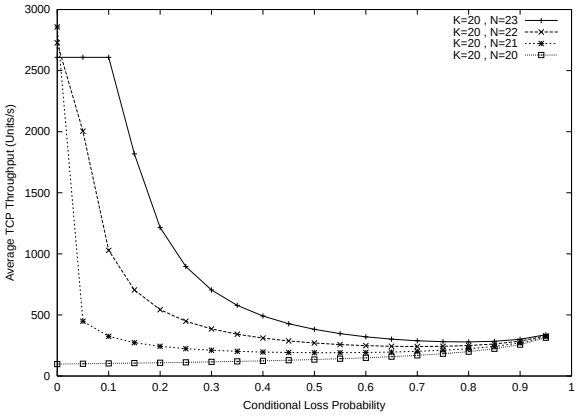


Figure 9.11: Model: $\bar{X}$ vs. β and N/K

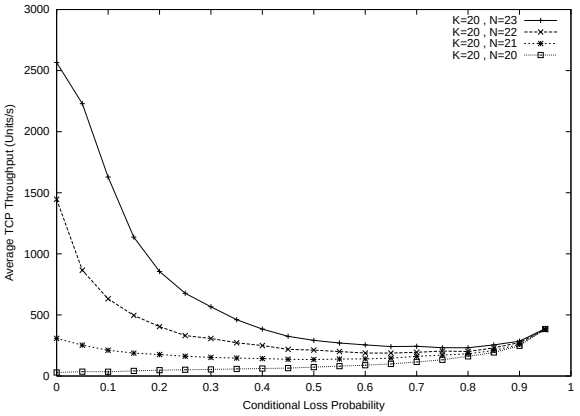


Figure 9.12: Simulation: $\bar{X}$ vs. β and N/K

calculated from (9.2), the expression of l. The other parameters of the model are taken as in the previous section. We see well that when the loss process is bursty (large β and small $\bar{\gamma}$), a large amount of FEC always gives the best performance. The difference in the performance between the different amounts of FEC is important for small bursts (small β). When burstiness increases, the throughput decreases drastically for the three FEC schemes we consider in the figure. A large amount of FEC helps the throughput to resist to small bursts but once these bursts become larger than the FEC capacity, the throughput deteriorates quickly. We also see that in the absence of FEC, the throughput improves a little when burstiness increases. We notice this improvement in the throughput for the three other FEC schemes at large β. As expected, all the curves converge to the same point when bursts become very large. At this point, the three amounts of FEC we consider in the figure have no benefit. Much FEC must be added to clean the link in this case. But, much FEC steals much bandwidth from TCP when burstiness decreases. A compromise must be done between much FEC to resist to bursts and a small amount of FEC to give better performance when burstiness decreases. This compromise exists if we want to use a static FEC scheme. But, one can think about using a dynamic FEC scheme where the FEC layer monitors the loss process on the wireless link and adapts the amount of FEC as a function of the rate of errors and their burstiness.

Now, we show in Figure 9.13 how the block size K can help TCP to resist to bursts of losses. We maintain the same amount of FEC ($N/K = 11/10$) and we vary K. Increasing K increases the number of redundant units in a TCP packet and thus helps it to resist to larger bursts. We also notice that a large block size still gives better performance when the length of bursts of losses is larger than the number of redundant units per packet. As we said before, this is due to the fast growth of the window in terms of units when large blocks are used. The problem with large blocks is that they require a long time to code/decode the redundant information. Again one can think about using a dynamic FEC scheme where the size of blocks changes with the

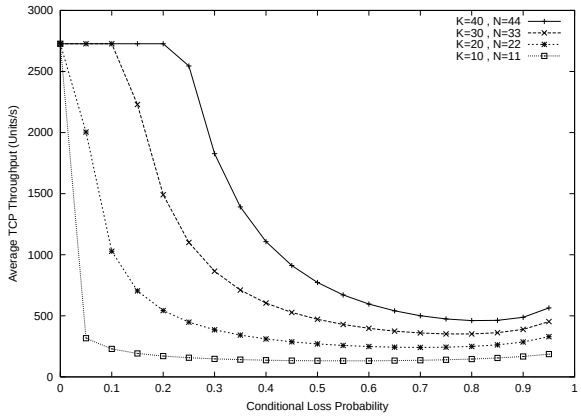


Figure 9.13: Model: $\bar{X}$ vs. β at constant N/K

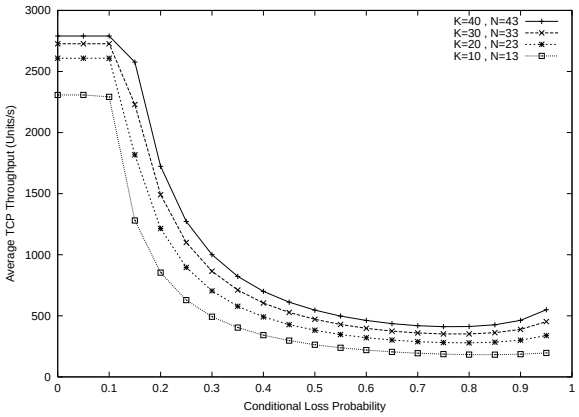


Figure 9.14: Model: $\bar{X}$ vs. β at constant R

burstiness of transmission errors.

The benefit of large packets is also illustrated in Figure 9.14. In this figure, we plot for the same number of redundant units per packet ($R = 3$), the variation of the throughput for different packet sizes. It is clear how a large packet gives better performance than a small one even though the amount of FEC is smaller. From Equation (9.7), increasing K for the same R decreases T, but this decrease is small compared to the gain we get from the increase in the window growth rate. In other words, the throughput in terms of packets/s deteriorates when we increase K at a constant R, but in terms of units/s it improves. Which mostly counts is the number of redundant units per packet rather than the total amount of redundancy. Of course, if we increase the number of redundant units per packet together with the packet size, we improve further the performance and this is exactly what we see in Figure 9.13.

9.4.3 Simulation results

We consider the SACK version of TCP [56] which is able to recover from a burst of packet losses without reducing its window multiple times and without resorting to timeout and slow start. This should give close results to the expression of the throughput we are using. The parameters of the network are not changed. l is set to 0.01 and K to 20. Our intention is to validate by simulation the analytical results we plotted in Figure 9.11. With these settings, Assumption 9.2 is satisfied for all the values of β we consider in the figure.

The simulation results are plotted in Figure 9.12. The curves in the figure show the same behavior as those in Figure 9.11. But, we see some mismatch at low burstiness. This is due to our assumption that a packet can only be lost by a single long burst not by multiple small and separate bursts of losses at the unit level. As one must expect, the simulation gives a lower throughput in this region given that we are overestimating T. This mismatch disappears at high correlation. For large β, $\bar{\gamma}$ is small and the probability to lose a packet due to separate bursts

becomes negligible. This makes our expression for T more precise.

9.5 Conclusions

In this chapter, we studied via mathematical analysis and simulations the interactions between the parameters of a FEC scheme and the performance of TCP congestion control. FEC is a promising approach for the improvement of the quality of noisy transmission media as wireless and satellite links. We showed that TCP throughput improves with the addition of FEC until a certain point where the noisy link becomes clean from TCP point of view. Any increase in FEC beyond this point is not beneficial. It reduces the available bandwidth and deteriorates the performance of TCP.

At a constant amount of FEC, we showed that the increase in the block size always results in an improvement of TCP performance. Large blocks contain more redundancy and are able to better resist to grouped errors. Moreover, large blocks permit TCP to increase faster its window after the occurrence of losses. Another result of our study is that it is possible to improve further the performance by opening multiple TCP connections to the same destination. With multiple connections, less FEC is required to clean a noisy link and the throughput of TCP can reach higher values. Concerning burstiness we showed that, even if the average error rate remains unchanged, the increase in burstiness reduces the efficiency of FEC and deteriorates the throughput of TCP. An addition of FEC can solve the problem but this FEC becomes unnecessary when burstiness disappears. We also found that an increase in block size improves the resilience to bursts of errors without any further addition of FEC. We can even improve the performance by increasing the block size without changing the number of redundant units per packet.

The main conclusion that we can derive from our study is that before the addition of any amount of FEC, the designers of wireless links have to choose the largest possible block size. Once the block size is chosen, they have to choose the amount of FEC as a function of the rate and burstiness of losses and this is in a way to clean the wireless link from the point of view of a single TCP connection. If the burstiness of errors frequently changes, a dynamic FEC mechanism could be envisaged to adapt the number of redundant units per packet as a function of the length of bursts.

Chapter 10

Conclusions and perspectives

Over the different chapters of this thesis, we investigated the performance of TCP congestion control. Such investigation is important given the central role played by the TCP protocol in the operation and the evolution of the Internet. Our study was based on modeling tools and the different results were validated via simulations and real measurements.

We adopted two distinct approaches in our study. These are the same two approaches used in the literature for the analysis and improvement of TCP congestion control. The first approach consists in studying the performance of the protocol on end-to-end basis without looking at the content of the network between the two TCP terminals. Our main objective was to find simple expressions for the throughput of a TCP connection that can cover the different ways with which the network reacts to the packets the connection transmits. With such expressions, one can decide on the different factors that impact the performance of TCP and that prohibit the protocol from achieving its objectives. These expressions also give insights on how to optimize the network and TCP mechanisms so as to get better performance. Another application of such expressions is to predict correctly the throughput of existing versions of TCP so as to establish some kind of fairness between TCP flows and non-TCP flows. Our work in this direction resulted in a general framework for TCP throughput calculation where we explained how the different mechanisms of TCP congestion control can be modeled and where we presented a general formula for TCP throughput that accounts for the different ways with which an IP network drops TCP packets (e.g., average time between drops, variation of time between drops). Our study showed the simplicity of existing models for TCP and the necessity of a general approach as the one we introduced. The benefits of our general formula compared to existing formulas for TCP throughput were proved with real measurements over the Internet. For example, one of our main results is that existing models for TCP make the simplistic assumption that the network drops packets in a deterministic way, so that they lead to an underestimation of the real throughput when the time intervals between drops vary.

Based on the end-to-end approach, we also presented a Markovian model useful for the calculation of TCP throughput on Internet paths where the loss rate of TCP packets fluctuates

according to a Markov chain. The advantage of a Markovian model on such paths is that it requires the identification of a smaller number of parameters than a general model that makes no Markovian assumptions. Finally, we addressed in a separate chapter the problem of calculation of TCP throughput when the receiver window limits the increase in the congestion window. We succeeded to calculate an explicit expression of the throughput when the loss rate of TCP packets within the network is homogeneous and memoryless. We also found for these particular networks the distribution of TCP window as well as a recurrent equation that gives all its moments. Approximations of TCP throughput were found in case of such limitation for more general loss models.

The second approach we adopted in this thesis is a network-specific approach where the performance of TCP is studied in some challenging environments. We considered the three environments that are supposed to be the most challenging for TCP: the large bandwidth-delay product network, the asymmetric network and the wireless network. For each network, we developed an analytical model for the evaluation of TCP performance. Our study showed some tuning problems with TCP mechanisms as well as with the solutions proposed in the literature to help the TCP connection. We used the results of our modeling to propose new mechanisms that improve the performance of TCP transfers. Our mechanisms were added to the `ns` simulator and they showed a gain in performance compared to existing ones.

This thesis presented some analysis of TCP performance as well as some mechanisms and guidelines to help the protocol. However, the problem of TCP congestion control is still open. Further analysis is still needed and some problems of the protocol still have to be solved. One should expect more problems to appear when new transmission media will be added to the Internet. Briefly, we mention some open problems (analysis and design) that need to be addressed. We start by the end-to-end approach. The most urgent problem seems to be the long time taken by the slow start phase especially on long delay links. The appropriate solution to this problem is most likely the estimation of network capacity and the spacing of packets. However, further study is required to find an appropriate mechanism for the estimation of the rate at which packets can be safely paced during slow start without overwhelming network buffers. Another interesting problem that one can address is the optimization of TCP congestion control parameters (window increase rate, window reduction factor) using our explicit expressions of the throughput. Such optimization problem can be combined with the ECN (Explicit Congestion Notification) proposal [61] which tolerates a less conservative window reduction at the TCP source. Recall that ECN provides the source with an explicit congestion signal when a congestion starts to form in the network and this is before the start of packet dropping. In the absence of ECN, there is no way to distinguish between a sever congestion, a transient congestion, an early congestion in a RED (Random Early Detection) buffer [65], etc. There is a consensus in the research community that a TCP source must divide its window by two whenever a packet is lost in the network.

Concerning our models for TCP throughput, our current work focuses on the refinement of

these models and on their applications. The introduction of the sub-linearity of window increase is one of the issues that we are actually investigating [27]. Another issue is the calculation of TCP throughput in case of window limitation for a more general loss process than the one we considered in Chapter 6, for example for a loss process modeled as a Markov Modulated Point Process [59]. Concerning the application of our results, we are looking at how the throughput of a TCP connection can be approximated on runtime without the need to consider the entire trace file of the connection. This requires the definition of some averaging algorithms to calculate on runtime the different functions of the process of congestion events (e.g., variance of time intervals between congestion events). Without loss of generality, one can use similar algorithms to those used by TCP for the calculation of the average and variance of round-trip time. Our idea is to use our general expression of TCP throughput in a TCP-friendly multimedia tool as TFRC (TCP-Friendly Rate Control) [63] and see what gain in performance we obtain. The other issue that we are actually working on in the application direction is to evaluate how much changing the congestion control policy of TCP changes the process of congestion events seen by the connection. This will tell us how much correct is the use of the process of congestion events seen by a non-TCP flow (e.g., an audio flow) to infer the process of congestion events that a TCP connection running on the same path will see. We believe that on paths where the rate of the non-TCP flow is small compared to the rate of the exogenous traffic and where multiple congested routers are crossed (e.g., on a long distance connection), the two processes will be close to each other since the moments of congestion will be independent of the transmission rate. We are also working on other applications of our analytical results. One idea is to calculate the functions of the process of congestion events for some particular networks, for example for networks that drop packets with a constant probability or for those that drop packets with a probability function of the window size of the TCP connection. Once calculated, these functions can be plugged in our general expression in order to find the exact throughput of TCP. Note that models for the network are important to evaluate the impact of a mechanism we add to the network to help TCP congestion control. Typical examples of network models are the ones used in [95, 97] to study how well TCP behaves with RED buffers. Another example is the model proposed in [48] for a set of RED routers and a set of TCP connections. The throughput of an individual TCP connection is calculated as a function of the probability that one of its packets is dropped within the network and the model is solved for the throughput of TCP, the packet drop probability in each router, the average queue length in each router, etc.

Consider now the network-specific approach. The mechanisms we proposed for the window increase during slow start and for the filtering of ACKs on the reverse path of a TCP connection still have to be tested in a real environment. Concerning the FEC scheme we studied in Chapter 9, one possible extension is to come up with an adaptive scheme that changes the amount of FEC and the bock size as a function of the way with which transmission errors occur on the wireless interface. In addition to that, one can always think about studying the performance of TCP

in other new environments, for example in a Differentiated Services network [52, 101] or in a multiple access medium (e.g., Dynamic Allocation Multiple Access protocol). We have started to develop analytical models for TCP congestion control in Differentiated Services networks. Preliminary results have already appeared in [17, 30]. Our objective is to show how well TCP connections are able to realize the throughput they subscribe in a network with different classes of traffic. For multiple access media allocating bandwidth to TCP connections in slots, one can imagine the study of the impact of the periodic availability of the path on the variation of round-trip time and on the burstiness of TCP traffic. We believe that the network-specific approach for the study of TCP congestion control will exist as long as new mechanisms and new transmission media are introduced into the Internet. Now, analytical models can change from one environment to another and as a function of the performance measure we want to calculate (e.g., throughput, duration of slow start, etc.). With the three environments we considered in this thesis, we gave an idea on how a network-specific analysis of TCP congestion control can be conducted.

Bibliography

[1] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Perforamnce of TCP Pacing", *IEEE INFOCOM*, Mar. 2000.

[2] M. Allman, "On the Generation and Use of TCP Acknowledgments", *ACM Computer Communication Review*, vol. 28, no. 5, pp. 4-21, Oct. 1998.

[3] M. Allman et al., " Ongoing TCP Research Related to Satellites", *RFC 2760*, Feb. 2000.

[4] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", *Internet Draft*, work in progress, Aug. 2000.

[5] M. Allman and A. Falk, "On the Effective Evaluation of TCP", *ACM Computer Communication Review*, vol. 29, no. 5, Oct. 1999.

[6] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", *RFC 2414*, Sep. 1998.

[7] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", *RFC 2488*, Jan. 1999.

[8] M. Allman, H. Kruse, and S. Ostermann, "An Application-Level Solution to TCP's Satellite Inefficiencies", *First International Workshop on Satellite-based Information Services (WOSBIS)*, Nov. 1996.

[9] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties", *ACM SIGCOMM*, Sep. 1999.

[10] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", *RFC 2581*, Apr. 1999.

[11] E. Altman, K. Avratchenkov, and C. Barakat, "TCP in Presence of Bursty Losses", *ACM SIGMETRICS*, Jun. 2000.

[12] E. Altman, K. Avranchenkov, and C. Barakat, "A stochastic model for TCP/IP with stationary random losses", *ACM SIGCOMM*, Aug. 2000.

[13] E. Altman, K. Avratchenkov, and C. Barakat, "TCP in Presence of Bursty Losses", *Performance Evaluation*, vol. 42, no. 2-3, pp. 129-147, Oct. 2000.

[14] E. Altman, K. Avrachenkov, C. Barakat, and P. Dube, "TCP over a Multi-State Markovian Path", *Conference on the Performance and QoS of Next Generation Networking (P&QNet)*, Nov. 2000.

[15] E. Altman, K. Avrachenkov, C. Barakat, and R. N. Queija, "State-dependent M/G/1 Type Queueing Analysis for Congestion Control in Data Networks", *CWI Report*, no. PNA-R0005, Jul. 2000.

[16] E. Altman, K. Avrachenkov, C. Barakat, and R. N. Queija, "State-dependent M/G/1 Type Queueing Analysis for Congestion Control in Data Networks", to appear in *IEEE INFOCOM*, Apr. 2001.

[17] E. Altman, C. Barakat, E. Laborde, P. Brown, and D. Collange, "Fairness Analysis of TCP/IP", *IEEE Conference on Decision and Control*, Dec. 2000.

[18] E. Altman, J. Bolot, P. Nain, D. Elouadghiri- M. Erramdani, P. Brown, and D. Collange, "Performance Modeling of TCP/IP in a Wide-Area Network", *IEEE Conference on Decision and Control*, Dec. 1995.

[19] M. Aron and P. Druschel, "TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control", *Rice Technical Report*, no. TR98-318.

[20] S. Asmussen and G. Koole, "Marked point processes as limits of Markovian arrival streams", *Journal of Applied Probability*, vol. 30, pp. 365-372, 1993.

[21] "The ATM Forum Traffic Management Specification Version 4.0", *ATM Forum Traffic Management AF-TM-0056.000*, Apr. 1996.

[22] F. Baccelli and P. Bremaud, "Elements of queueing theory: Palm-Martingale calculus and stochastic recurrences", *Springer-Verlag*, 1994.

[23] F. Baccelli and D. Hong, "TCP is Max-Plus Linear", *ACM SIGCOMM*, Aug. 2000.

[24] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", *International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[25] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance", *ACM MOBICOM*, Sep. 1997.

[26] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz, "A comparison of Mechanisims for Improving TCP Performance over Wireless Links", *ACM SIGCOMM*, Aug. 1996.

[27] C. Barakat, "TCP modeling and validation", *to appear in IEEE Network.*

[28] C. Barakat and E. Altman, "Analysis of TCP with Several Bottleneck Nodes", *IEEE GLOBECOM*, Dec. 1999.

[29] C. Barakat and E. Altman, "Performance of Short TCP Transfers", *Networking 2000 (Performance of Communications Networks)*, May 2000.

[30] C. Barakat and E. Altman, "A Markovian Model for TCP Analysis in a Differentiated Services Network", *International Workshop on Quality of future Internet Services (QofIS)*, Sep. 2000.

[31] C. Barakat and E. Altman, "On ACK Filtering on a Slow Reverse Channel", *International Workshop on Quality of future Internet Services (QofIS)*, Sep. 2000.

[32] C. Barakat and E. Altman, "Bandwidth tradeoff between TCP and link-level FEC", *to appear in IEEE ICN*, Jul. 2001.

[33] C. Barakat and E. Altman, "Analysis of the Phenomenon of Several Slow Start Phases in TCP", *ACM SIGMETRICS*, extended abstract, Jun. 2000.

[34] C. Barakat, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network : A Survey", *IEEE Communications Magazine*, vol. 38, no. 1, pp. 40-46, Jan. 2000.

[35] C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Improving TCP/IP over Geostationary Satellite Links", *IEEE GLOBECOM*, Dec. 1999.

[36] R. Bellman, "Introduction to matrix analysis", *McGraw-Hill*, New York, 1960.

[37] J. Bennett, C. Partridge, and N. Shectman, " Packet reordering is not pathological network behavior", *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 789-798, Dec. 1999.

[38] S. Biaz and N. H. Vaidya, “Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result”, *International Conference on Computer Communications and Networks (IC3N)*, Oct. 1998.

[39] T. Bonald, "Stabilité des systèmes dynamiques à événements discrets, application au contrôle de flux dans les réseaux de télécommunications”, *PhD thesis*, Université de Nice-Sophia Antipolis, Oct. 1999.

[40] A. Borovkov, "Ergodicity and Stability of Stochastic Processes", *Wiley*, 1998.

[41] R. Braden, "Requirements for Internet Hosts – Communication Layers", *RFC 1122*, Oct. 1989.

[42] B. Braden, et al.,“Recommendations on Queue Management and Congestion Avoidance in the Internet”, *RFC 2309*, Apr. 1998.

[43] A. Brandt, “The stochastic equation $Y_{n+1} = A_n Y_n + B_n$ with stationary coefficients”, *Advances in Applied Probability*, vol. 18, pp. 211-220, 1986.

[44] A. Brandt. P. Franken, and B. Lisek, “Stationary stochasitic models”, *Wiley*, 1990.

[45] L. Brakmo and L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet”, *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465-1480, Oct. 1995.

[46] P. Brown, "Resource sharing of TCP connections with different round trip times", *IEEE INFOCOM*, Mar. 2000.

[47] K. Brown and S. Singh, “M-TCP: TCP for Mobile Cellular Networks”, *ACM Computer Communication Review*, vol. 27, no. 5, pp. 19-43, Oct. 1997.

[48] T. Bu and D. Towsley, "Fixed Point Approximation for TCP behavior in an AQM Network", *UMass CMPSCI Technical Report*, no. 00-43, Jul. 2000.

[49] H. Chaskar, T. V. Lakshman, and U. Madhow, “On the design of interfaces for TCP/IP over wireless”, *IEEE MILCOM*, Oct. 1996.

[50] D. Chiu and R. Jain, “Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks”, *Journal of Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1-14, Jun. 1989.

[51] A. Chockalingam, M. Zorzi, and R.R. Rao, “Performance of TCP on Wireless Fading Links with Memory”, *IEEE ICC*, Jun. 1998.

[52] D. Clark and W. Fang, “Explicit Allocation of Best Effort Packet Delivery Service”, *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362-373, Aug. 1998.

[53] R. Durst, G. Miller, and E. Travis, “TCP Extensions for Space Communications”, *ACM MOBICOM*, Nov. 1996.

[54] L. Eggert, J. Heidemann, and J. Touch, "Effects of Ensemble-TCP", *ACM Computer Communication Review*, vol. 30, no. 1, pp. 15-29, Jan. 2000.

[55] B. R. Elbert, “The Satellite Communication Applications Handbook”, *Artech House*, Boston, London, 1997.

[56] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP”, *ACM Computer Communication Review*, vol. 26, no. 3, pp. 5-21, Jul. 1996.

[57] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, *RFC 2068*, Jan. 1997.

[58] V. Firoiu and M. Borden, "Queue Management for Congestion Control", *IEEE INFOCOM*, Mar. 2000.

[59] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook", *Performance Evaluation*, vol. 18, no.2, pp. 149-171, 1993.

[60] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic", *ACM Computer Communication Review*, vol. 21, no. 5, pp. 30-47, Oct. 1991.

[61] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10-23, Oct. 1994.

[62] S. Floyd and K. Fall, "Promoting the Use of End-To-End Congestion Control in the Internet", *IEEE/ACM Transactions in Networking*, vol. 7, no. 4, pp. 458-472, Aug. 1999.

[63] S. Floyd, M. Handley and J. Padhye, "Equation-based congestion control for unicast applications: the extended version", *ACM SIGCOMM*, Aug. 2000.

[64] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", *RFC 2582*, Apr. 1999.

[65] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413, Aug. 1993.

[66] E.N. Gilbert, "Capacity of a burst-noise channel", *Bell Systems Technical Journal*, Sep. 1960.

[67] P. Glasserman and D. D. Yao, "Stochastic vector difference equations with stationary coefficients", *Journal of Applied Probability*, Vol. 32, pp. 851-866, 1995.

[68] R. Goyal, R. Jain, S. Kota, M. Goyal, S. Fahmy, and B. Vandalore, "Traffic Management for TCP/IP over Satellite-ATM Networks", *IEEE Communications Magazine*, vol. 37, no. 3, Mar. 1999.

[69] E. Hashem, "Analysis of random drop for gateway congestion control", *MIT*, no. TR-465, 1989.

[70] T. Henderson and R.H. Katz, "Transport Protocols for Internet-Compatible Satellite Networks", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 2, pp. 326-344, Feb. 1999.

[71] T. Henderson, E. Sahoria, S. McCanne, and R. H. Katz, "Improving Fairness of TCP Congestion Avoidance", *IEEE GLOBECOM*, Nov. 1998.

[72] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *ACM SIGCOMM*, Aug. 1996.

[73] J.J. Hunter, "On the moments of Markov renewal processes", *Advances in Applied Probability*, vol. 1, pp. 188-210, 1969.

[74] P. Hurley, J. Y. Le Boudec, and P. Thiran, "A Note on the Fairness of Additive Increase and Multiplicative Decrease", *ITC-16*, Jun. 1999.

[75] V. Jacobson, "Congestion avoidance and control", *ACM SIGCOMM*, Aug. 1988.

[76] V. Jacobson, "Compressing TCP/IP Headers for Low-speed Serial Links", *RFC 1144*, Feb. 1990.

[77] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance", *RFC 1323*, May 1992.

[78] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks", *ACM Computer Communication Review*, . vol. 19, no. 5, pp. 56-71, Oct. 1989.

[79] S.H. Kang and D.K. Sung, "A Markovian arrival process (MAP) modeling for superposed ATM traffic", manuscript.

[80] L. Kleinrock, "Queueing Systems", *Wiley*, 1975.

[81] C. Knessl, B. Matkowsky, Z. Schuss, and C. Tier, "Asymptotic Analysis of a state-dependent M/G/1 queueing system", *SIAM Journal on applied Mathematics*, vol. 46, no. 3, pp. 483-505, 1986.

[82] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link", *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 485-498, Aug. 1998.

[83] A. Kumar and J. Holtzman, "Performance Analysis of Versions of TCP in a Local Network with a Mobile Radio Link", *Sadhana: Indian Academy of Sciences Proceedings in Engg. Sciences*, Feb. 1998.

[84] T.V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss", *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336-350, Jun. 1997.

[85] T. V. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance", *IEEE INFOCOM*, Apr. 1997.

[86] T.V. Lakshman, A. Neidhardt, and T.J. Ott, "The Drop from Front Strategy in TCP over ATM and its Interworking with other Control Features", *IEEE INFOCOM*, Mar. 1996.

[87] D. Lin and R. Morris, "Dynamics of Random Early Detection", *ACM SIGCOMM*, Sep. 1997.

[88] J. Martin, A. Nilsson, and I. Rhee, "The Incremental Deployability of RTT-Based Congestion Avoidance for High Speed TCP Internet Connections," *ACM SIGMETRICS*, Jun. 2000.

[89] L. Massoulie and J. Roberts, "Bandwidth sharing: objectives and algorithms", *IEEE INFOCOM*, Mar. 1999.

[90] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", *ACM SIGCOMM*, Aug. 1996.

[91] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", *RFC 2018*, Oct. 1996.

[92] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", *ACM Computer Communication Review*, vol. 27, no. 3, pp. 67-82, Jul. 1997.

[93] K.S. Meier-Hellstern, "A fitting algorithm for Markov-modulated Poisson processes having two arrival rates", *European Journal of Operational Research*, vol. 29, pp. 370-377, 1987.

[94] I. Minei and R. Cohen, "High-Speed Internet Access Through Unidirectional Geostationary Satellite Channels", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 2, pp. 345-359, Feb. 1999.

[95] A. Misra and T. Ott, "The window distribution of idealized TCP congestion avoidance with variable packet loss", *IEEE INFOCOM*, Mar. 1999.

[96] V. Misra, W.-B. Gong, and D. Towsley, "Stochastic differential equation modeling and analysis of TCP-windowsize behaviour", *Performance*, Oct. 1999.

[97] V. Misra, W.-B. Gong, and D. Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with Application to RED", *ACM SIGCOMM*, Aug. 2000.

[98] R. Morris, "Scalable TCP Congestion Control", *IEEE INFOCOM*, Mar. 2000.

[99] J. Nagle, "Congestion control in IP/TCP internetworks", *RFC 896*, Jan. 1984.

[100] M.F. Neuts, "Structured stochastic matrices of M/G/1 type and their applications", *Marcel Dekker*, New York, 1989.

[101] K. Nichols, V. Jacobson, and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", *Internet Draft*, work in progress, May 1999.

[102] The LBNL Network Simulator, *ns*, http://www.isi.edu/nsnam/ns/

[103] N.C. Oguz and E. Ayanoglu, "Performance Analysis of Two-Level Forward Error Correction for Lost Cell Recovery in ATM Networks", *IEEE INFOCOM*, Apr. 1995.

[104] T. Ott, J. Kemperman, and M. Mathis, "The stationary behavior of ideal TCP congestion avoidance", manuscript, Aug. 1996.

[105] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: a Simple Model and its Empirical Validation", *ACM SIGCOMM*, Aug. 1998.

[106] J. D. Parsons, "The Mobile Radio Propagation Channel", *Pentech Press*, London, 1992.

[107] V. Paxson, "End-to-End Internet Packet Dynamics", *ACM SIGCOMM*, Sep. 1997.

[108] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer", *Internet Draft*, work in progress, Apr. 2000.

[109] L. Peterson and B. Davie, "Computer Networks: a system approach", *Academic Press*, 2000.

[110] M. Posner," Single-server queues with service time dependent on waiting time", *Operations Research*, vol. 21, pp. 610-616, 1973.

[111] J. Postel, "Transmission Control Protocol", *RFC 793*, Sep. 1981.

[112] K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", *RFC 2481*, Jan. 1999.

[113] T.S. Rappaport, "Wireless Communications", *IEEE Press*, New York, 1996.

[114] L. Rizzo, "Effective erasure codes for reliable computer communication protocols", *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24-36, Apr. 1997.

[115] S. Sahu, P. Nain, D. Towsley, C. Diot, and V. Firiou, "On Achievable Service Differentiation with Token Bucket Marking for TCP", *ACM SIGMETRICS*, Jun. 2000.

[116] N. Samaraweera and G. Fairhurst, "Reinforcement of TCP/IP Error Recovery for Wireless Communications", *ACM Computer Communication Review*, vol. 28, no. 2, pp. 30-38, Apr. 1998.

[117] S. Savari and E. Telatar, "The Behavior of Certain Stochastic Processes Arising in Window Protocols", *IEEE GLOBECOM*, Dec. 1999.

[118] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP Buffer Tuning", *ACM SIGCOMM*, Sep. 1998.

[119] N. Shacham and P. McKenney, “Packet Recovery in High-Speed Networks Using Coding and Buffer Management”, *IEEE INFOCOM*, Jun. 1990.

[120] B. Suter, T.V. Lakshman, D. Stiliadis, and A.K. Choudhary, “Design Considerations for Supporting TCP with Per-flow Queueing”, *IEEE INFOCOM*, Mar. 1998.

[121] W. Stevens, “TCP Slow-Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, *RFC 2001*, Jan. 1997.

[122] J. Stone and C. Partridge, "When the CRC and TCP Checksum Disagree", *ACM SIGCOMM*, Sep. 2000.

[123] K. Thompson, G.J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics", *IEEE Network*, vol. 11, no. 6, pp. 10-23, Nov. 1997.

[124] H.C. Tijms," Stochastic Models — An Algorithmic Approach", *Wiley*, Chichester, 1994.

[125] V. Visweswaraiah and J. Heidemann, “Improving Restart of Idle TCP Connections”, *University of Southern California Technical Report*, no. 97-661, Nov. 1997.

[126] J. Walrand, “An introduction to queueing networks”, *Prentice Hall*, 1988.

[127] Z. Wang and J. Crowcroft, "A new congestion control scheme: Slow start and search (tri-s)", *ACM Computer Communication Review*, vol. 21, no. 1, pp. 32-43, Jan. 1991.

[128] L. Zhang, S. Shenker, and D.D. Clark, “Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic”, *ACM SIGCOMM*, Sep. 1991.

[129] Y. Zhang, D. DeLucia, B. Ryu, and S. Dao, “Satellite Communications in the Global Internet: Issues, Pitfalls, and Potential”, *INET*, Jun. 1997.

Résumé

On étudie dans cette thèse les performances des mécanismes de contrôle de congestion du protocole TCP. Ces mécanismes sont très importants pour la stabilité de l'Internet étant donné le grand volume du trafic transporté par TCP. On développe pour ce but plusieurs modèles analytiques. Nos modèles sont divisés en deux groupes : les modèles de bout-en-bout et les modèles qui considèrent les caractéristiques et les mécanismes du réseau. L'objectif des modèles de bout-en-bout est de trouver des expressions simples pour le débit moyen d'une connexion TCP de longue durée. On essaie de garder nos modèles les plus généraux possibles pour qu'ils puissent couvrir les différentes manières avec lesquelles le réseau rejette les paquets de la connexion TCP. En utilisant des techniques de la théorie de processus stochastiques, on trouve des expressions explicites pour le débit moyen de TCP qu'on valide avec des mesures sur l'Internet. Concernant le deuxième groupe de modèles, l'objectif est d'étudier les performances des transferts TCP dans des environnements difficiles au protocole. On considère les trois environnements montrés dans la littérature comme étant les plus difficiles au protocole : l'environnement ayant un grand produit délai-bande passante, l'environnement ayant une importante asymétrie de bande passante entre le chemin d'aller et le chemin de retour, et l'environnement sans fil où les paquets TCP sont le plus souvent perdus pour une autre raison que la congestion. Pour chaque environnement, on propose un modèle analytique ainsi qu'un ensemble de mécanismes et de solutions pour aider le protocole TCP dans son fonctionnement. Les résultats de notre deuxième groupe de modèles sont validés par des simulations sur `ns`. La thèse contient aussi une présentation des différents travaux qui ont été effectués sur les mécanismes de contrôle de congestion de TCP depuis leur introduction dans l'Internet.

Mots-clés: Internet, contrôle de congestion, TCP, modélisation, processus stochastiques, proposition de mécanismes et de directions, expérimentation, simulation

Abstract

We study in this thesis the performance of the congestion control mechanisms of the TCP protocol. These mechanisms are very important for the stability of the Internet given the huge amount of the TCP traffic. For the purpose of our study we develop some analytical models. Our models are divided into two main groups: the end-to-end models and the network-specific models. The objective of the end-to-end models is to come up with simple expressions of the throughput of a long-life TCP connection. We try to keep our models as general as possible so that they cover the different ways with which the network drops the packets of the TCP connection. Using techniques from the theory of stochastic processes, we find explicit expressions for TCP throughput that we validate via measurements over the Internet. Concerning the network-specific models, our objective is to study the performance of TCP in challenging environments. We focus on the three environments considered in the literature as the most challenging for TCP: the large bandwidth-delay product environment, the asymmetric-bandwidth environment, and the wireless environment. For each environment, we propose an analytical model as well as some mechanisms and guidelines to improve the performance of TCP transfers. The different results of our second group of models are validated via `ns` simulations. The thesis also contains a presentation of the different works on TCP congestion control mechanisms since their introduction into the Internet.

Keywords: Internet, congestion control, TCP, modeling, stochastic processes, proposition of mechanisms and guidelines, experimentation, simulation