BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPT. OF TELECOMMUNICATIONS AND MEDIA INFORMATICS

A PERFORMANCE ANALYSIS OF SOME TRAFFIC CONTROL TECHNIQUES IN TCP/IP NETWORKS

Tuan Anh Trinh

Ph.D. Dissertation

Supervised by

Dr. Sándor Molnár and Dr. László Gefferth
High Speed Networks Laboratory
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics

Budapest, Hungary
2004

Tuan Anh Trinh
High Speed Networks Laboratory
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics[1]

[1]The reviews and the minutes of the Ph.D. Defense are available from the Dean's Office.

To my family

Table of Contents

List of Tables

List of Figures

Abstract

Traffic control techniques, by keeping the rapidly growing Internet traffic within control, are indispensable and play a crucial role in the success of the Internet. The aim of this dissertation is to provide a comprehensive performance analysis of some key traffic control techniques in TCP/IP networks. First, characterization of some metrics of TCP is presented. Next, a novel approach to model TCP traffic is provided. In addition, the rate control and parameter setting of TCP is analyzed from a game-theoretic point of view. Finally, a comprehensive performance analysis of one of the most well-known queue management schemes, the Random Early Detection scheme, is presented.

Acknowledgements

Throughout my graduate study, I have been fortunate enough to have the help and support of a large number of people. First of all, I would like to express my deep gratitude to Dr. Sándor Molnár for his kind encouraging guidance and support. Without his help this dissertation would have never been completed.

My special thanks are due to Prof. László Györfi for helping me in many ways throughout the years and Tamás Éltető for guiding me at the beginning when I was an undergraduate student. I owe them a depth of gratitude.

All my research work has been done in the framework of teletraffic research at High Speed Networks Laboratory (HsnLab), Dept. of Telecommunications and Media Informatics, Budapest University of Technology and Economics (TMIT/BUTE). I would like to thank Dr. Tamás Henk, head of the laboratory, for his support. I also wish to acknowledge the help that I received from those people at TMIT/BUTE during the years. It was my pleasure working together with them.

I would also like to thank Prof. Srinivasan Keshav (Stanford University, USA), Dr. Polly Huang (ETH Zürich, Switzerland), Dr. Miklós Telek, Dr. Attila Vidács, and Dr. András Veres (BUTE) for their helpfulness and fruitful advice contributing to my research work.

Finally, I would most like to thank my parents for their continual encouragement and support throughout the years. This work is dedicated to them.

Budapest,
July 2004

Trinh Anh Tuan

Chapter 1

Introduction

Starting from a relatively small network used by a relatively small research community in the United States, the Internet is now used by millions of people all over the world and this number is growing increasingly over the years. The success of the Internet, in which the TCP/IP protocol suite plays an important role, is based on its open, flexible but robust architecture.

It has not all been a happy story, nevertheless. An incident [Nag84] happened during the early growth phase of the Internet in mid 1980s brought the Internet down. The incident (technically called *congestion collapse*) was later explained by the lack of attention to the dynamics of packet forwarding. The lesson learnt from this incident is that we need to introduce some *transmission control* mechanisms into the design of the Internet. In this respect, the original fix to the congestion collapse of the Internet was provided by Van Jacobson in [Jac88] and the Transmission Control Protocol (TCP) was born. The basic idea behind TCP congestion control is to control network load by having sources adjust their rates according to the level of congestion in the network. Since then, a number of improvements have been added to the original TCP implementation, this basic idea remained unchanged.

In addition, traffic in the Internet is composed of flows with different nature and different characteristics, as more and more new IP-based applications are brought into existence. Some of them are congestion-aware and some are not. As a consequence, end-to-end congestion control algorithms such as those in TCP are not enough to prevent congestion in the Internet, and they must be supplemented by control mechanisms inside the network. Since routers are the common points for all flows, it is reasonable to detect and control congestion at these points, at least globally. The Drop Tail buffer management scheme does little in this respect. To face this problem, a number of Active Queue Management (AQM) schemes [FloJa93, FGS01, FKSS99, ALLY01, HMTG01] were introduced that can efficiently manage the buffers at the routers in order to avoid congestion, and in some cases, to guarantee fairness between competing flows.

From what have been discussed so far, it is without doubt that the the traffic control schemes play a crucial role in the success of the Internet. However, these schemes are still not well understood. As a result, there is a genuine need to have a better understanding of

their performance as well as their impact on the performance of the Internet as a whole in order to design more efficient traffic control mechanisms for the Internet.

The dissertation is organized around the above arguments. In the first part of the dissertation, we present research results from different perspectives on the modelling and analysis of the TCP protocol. The perspectives include the metrics of TCP, modelling of TCP traffic and the analysis of the rate control as well as the parameter setting problems of TCP from a game-theoretic point of view. The second part of the thesis presents a comprehensive analysis of one of the most promising AQM schemes, the Random Early Detection (RED) scheme, and proposes a novel scheme to improve RED performance.

1.1 New Insights in TCP Traffic Modelling and Analysis

In what follows, we summarize the key previous work in the field of TCP traffic modelling and analysis, introduce our motivation for research, and our contributions to this area.

Our research methodology is to apply different mathematical modelling techniques, network simulations and network measurements. We also introduce a number of new concepts and propose a number of new algorithms. When investigating the proposals, we combine the techniques mentioned above whenever possible to validate the results and to gain a better insight into the problem.

1.1.1 Previous Work on TCP Traffic Modelling and Analysis

There exists a substantial literature on the modelling and analysis of TCP. Extensive measurements and analysis of TCP have been carried out over the past few years, [LTWW93, Pax94, Pax97, RFC2525]. A lot of lessons have been learnt from these measurements and analysis. For example, it is now widely accepted that Internet traffic (in which TCP plays a crucial role)is self-similar in nature, [LTWW93], contradicting the Poisson assumption in the past. We have also learnt from the analysis about the pitfalls and many implementation problems of TCP, [RFC2525]. However, as new applications are being built upon TCP and new versions of TCP are being proposed for specific networks, more measurement and analysis is still need to have a better insights into the dynamics of the new/proposed networks. In addition, given the protocol, we want to *predict* the performance of the network and keep the network manageable and analytically tractable. In order to do that, we need to build *analytical models* for TCP. Basically, TCP modelling can be found in two main levels: packet level and flow (fluid) level. One of the motivations for the packet level approach is the possibility of applying existing discrete-time models [Kum98, PFTK98]. Respectively, the motivation for fluid level model is the possibility of applying existing continuous-time (control-theoretic) models, [Low03, MGT00, OKM96], to name a few. In both approaches, good points have been addressed and important, subtle results have been achieved. In [OKM96], Ott *et al* used stochastic differential equations to model TCP behavior and first suggested the well-known *square-root* formula. Padhye *et al* in [PFTK98] extend the model

in [OKM96] to capture Time Out. This model is widely accepted as one of the most accurate models for TCP Reno (in the case of bulk data transfer). We can also mention here the chaotic nature of TCP and its ability to propagate long range dependance in the Internet as suggested and examined in [VeBo00, VMKV00]. However, as TCP modelling is application-sensitive, a general purposed TCP model that is precise, yet simple, is still unavailable. This makes the TCP modelling task still very challenging and motivates us on our research.

1.1.2 Contributions to TCP Traffic Modelling and Analysis

In this dissertation, we investigate the properties of the TCP protocol in three different perspectives. The perspectives include the metrics of TCP, modelling of TCP traffic, and game-theoretic analysis of the rate control and parameter setting of TCP. Chapter 2 introduces new algorithms to measure and to evaluate new metrics of TCP. Chapter 3 proposes a novel approach to model TCP traffic. Chapter 4 analyzes the rate control and parameter setting of TCP from a game-theoretic perspective.

New algorithms to measure metrics of TCP

In order to understand the dynamics of TCP, we first need to know how it *actually* works. In Chapter 2, new metrics and novel algorithms are introduced to characterize the dynamics of TCP traffic in different perspectives. It is sometime not obvious how to measure some important metrics of TCP (such as the congestion window) from simulation as well as in real measurement. In fact, from time to time, we need to *approximate* the theoretically-defined metrics in practice. Moreover, as new theoretical approach requires *new kind of statistics and metrics*, it is essential to know how to measure these new metrics firstly for the sake of validation. Secondly, the new metrics themselves provide us new perspective and insight into the dynamic of TCP.

In Chapter 2, we introduce the *virtual queue* metric that characterizes the fluctuation of the number of packets in forward direction of a TCP connection over time. While the congestion window represents the traffic control at *end-point*, the virtual queue shows the impact of the congestion window on the *network*. We also design a novel algorithm to measure this metric. In addition, we discuss how this metric can reveal important information on the network dynamics.

We introduce and design a novel algorithm to detect the changes of states of TCP during a connection. The benefits of the algorithm are twofold. First, it helps us to have a better insight into how the TCP actually works. Secondly, the detection of the states TCP helps us to reveal important statistics of TCP.

Being able to detect the state changes of TCP, we are now possible to design and implement algorithms to measure important state-based metrics of TCP. The metrics include:

- The sojourn time distribution at each state during a TCP connection.
- The jumping probabilities from one state to another state during a connection.

- The distribution of the number of packets sent in each time slot (RTT).

These state-based metrics reveal the dynamics of TCP according to its states. In addition, the metrics help us collect the statistics to validate our new TCP model presented in Chapter 3.

A new unified model for TCP

Previous work on modelling of TCP is mainly based on studying the dynamics of the congestion window process. Although important results have been achieved, due to the reasons discussed above, a generic model for TCP that is simple and precise is still unavailable.

In Chapter 3, we propose a novel approach to model TCP traffic. In contrast to the congestion window based approach, we introduce a new perspective into the modelling paradigm of TCP. We study the dynamics of the *states* of the TCP itself in order to model it. We investigate the sojourn time distributions at the states as well as the jumping probabilities between the states and use the results achieved from the investigation to build a model to estimate TCP throughput. The main advantages of our approach are that (1) it provides a *unified* model for all well-known versions of TCP because they share the same logical set of states, and (2) the number of states is *significantly* reduced (in contrast with the number of possible values of the congestion window). Reducing the number of states in a Markov model makes it more computationally feasible and more analytically tractable, and thus, more desirable.

We propose a unified model for some well-known versions of TCP based on the Discrete-time Batch Markovian Arrival Process (D-BMAP). The basic idea behind our model is that we try to mimic inherent operation of TCP in order to model it. During a connection, TCP stays in any of the following states: Slow-Start, Congestion Avoidance, Fast Recovery, Exponential Back-off. TCP can jump from one state to another state in response to external events such as packet loss or Time Out. We consider how much time TCP stays in each state and the distribution of time elapsed at each state. We then consider the jumping probability from one state to another state. This inherent operation of TCP suggests us the use of the D-BMAP process (originally introduced and examined in detail in [BloCa95]). The idea of D-BMAP can be traced back to M. Neuts' work in [Neu81]. The states of the background process (modulating process) are the states of TCP itself (i.e. Slow Start, Congestion Avoidance, Loss Recovery and Time Out).

We introduce a new concept to characterize a TCP connection, namely the TCP characterization matrix, which is the matrix that characterizes the modulating Markov chain. we show how, under certain assumptions, the elements of the TCP characterization matrix can be expressed and computed.

Based on the model, we propose a generic formula to estimate the average bandwidth of a long TCP connection. The model is validated by simulation against different versions of TCP such as TCP Reno, TCP Tahoe, TCP NewReno and TCP SACK.

A game-theoretic analysis of TCP Vegas

With the emergence of very large bandwidth-delay product networks such as the transatlantic link with a capacity in the range of 1 Gbps - 10 Gbps, new transport protocols have been proposed to better utilize the network in these circumstances. A promising proposal is the FAST TCP, [JWL04]. Since the design of FAST TCP is heavily based on the design of TCP Vegas, there is a need to reconsider the benefits as well as the drawbacks of TCP Vegas in order to have an insight into the performance and possible deployment of FAST TCP in the future Internet. In Chapter 4, we analyze the rate control problem of a general TCP Vegas network from the game-theoretic point of view. We also show the impact of the parameter setting of TCP Vegas and FAST TCP on their performance by using a game-theoretic approach.

We first consider the rate control problem of a general TCP Vegas network as a non-cooperative game. One of the key questions in a non-cooperative flow control game in general, and our game in particular, is whether the network converges to (or settles at) an equilibrium point, such that no player can increase its payoff by adjusting its strategy unilaterally. In the game-theory terminology such a point is called a *Nash equilibrium*. The Nash equilibrium in our game also reflects the *balance* of the gain and the cost for each player as well as for the network as a whole. A non-cooperative game may have no Nash equilibria (in its pure strategy space), multiple equilibria, or a unique equilibrium. We then consider the parameter setting of TCP Vegas. As described in [BMP94], TCP Vegas tries to maintain the number of backlogged packets in the network between α and β (the parameters of TCP Vegas). We consider the case when N TCP Vegas flows sharing a single bottleneck link with capacity μ and with a buffer of size B. The TCP Vegas flows (the *players*) are assumed to be selfish (and greedy) - they all try to increase the number of their backlogged packets in the network. We are interested in a situation (i.e. a parameter setting, if at all exists) from where no player would deviate.

We prove that there exists a unique Nash equilibrium (in its pure strategy space) for the TCP Vegas rate control game.

We model the parameter setting problem of TCP Vegas as a static and noncooperative game. We demonstrate how selfish behaviour of users can make the all TCP Vegas-network vulnerable to congestion.

We also extend our analysis to FAST TCP case. We show how FAST TCP can be considered as a "faster" Vegas TCP and apply the analysis of the Vegas TCP case. Our analysis shows the all FAST TCP network is vulnerable to congestion and this poses a serious threat to the wide deployment of FAST TCP in future Internet.

1.2 Analysis of the Random Early Detection Mechanism

In the second part of the dissertation a comprehensive performance analysis of the Random Early Detection mechanism is provided. In Chapter 5, we revisit a number of concepts and found that previous thinking needs to be changed. Among these is the changed view that

actually RED does not provide the capability to apportion loss between flows. We highlight the problems associated with the tuning of RED and propose a novel scheme to enhance RED performance.

1.2.1 Previous Work on RED Performance Analysis

There exists a substantial literature on performance analysis of RED. We would divide it into two classes. The first class largely deals with analyzing and configuring RED, while keeping the algorithm intact, [MBB00, CJOS00, MGT00, HMTG01]. The second class considers how to change the original RED to have better performance, [FKSS99, FGS01, ZhAt00, LiMo97, AOMC01, WyZu02].

Despite the fact that extensive research has been devoted to performance analysis of RED and many publications have highlighted various aspects of this issue, the question of how to configure the parameters of RED for optimal performance is still open. In addition, the impact of RED mechanism on different issues of Internet performance (such as fairness) is still not clear and requires more clarification and analysis.

1.2.2 Contributions to RED Performance Analysis

In Chapter 5, we first revisit the proportional loss property of RED. Loosely speaking, the proportional loss property means that the fraction of marked packets for each connection is proportional to that connection's share of the bandwidth. RED is claimed to possess this property [FloJa93]. In addition, proportional loss is widely adopted in the fairness analysis of RED, [LiMo97], [HBT99]. Since TCP flows account for a large portion of Internet traffic, TCP arrivals are mainly of interest. We prove that packet losses between flows in TCP/RED networks is non-proportional by applying the ASTA properties.

Some solutions have been proposed to overcome the difficulties of tuning RED parameters. We can mention here the Adaptive RED of Feng [FKSS99] and the Adaptive RED of Floyd [FGS01]. These proposals try to change the max_p parameter to adapt to the changing condition of incoming traffic. My argument was that the inflexibility of RED not only lies in its max_p parameter but also on the EWMA mechanism. As a result, RED parameters can be tuned in an on-line manner by making the EWMA adaptive to the changing condition of the incoming traffic. We propose a new AQM scheme (Fuzzy RED) to alleviate the inflexibility of RED tuning.

We discuss the advantage as well as the disadvantage of the Exponential Weighted Moving Average (EWMA) mechanism in RED both from theoretical and practical point of view.

We propose the use of Fuzzy EWMA to overcome the inflexibility of RED. We enhance RED mechanism by replacing EWMA mechanism with Fuzzy EWMA mechanism.

We carry out a comparative performance analysis of different adaptive RED schemes by simulations. We investigate different scenarios: stationary cases, multiple sources with

different RTT, different number of flows, dynamically changing number of flows. The simulations show that, in the case of a high workload and a high level of variation, fuzzy RED, by tracking system variation in an on-line manner, improves RED performance in a number of important router-based metrics like packet loss rate, average queueing delay, link utilization, and global power.

Chapter 2

TCP Metrics Measurements

In order to understand the dynamics of TCP, we first need to know how it *actually* works. It is sometime not obvious how to measure some metrics of TCP from simulation as well as in real measurement. In fact, from time to time, we need to *approximate* the theoretically-defined metrics in practice.

One of the most important metrics of TCP is the congestion window. The congestion window (*cwnd*) is a sender's variable. More precisely, it is an inner variable of the operating system at the sender. Together with the advertised window (a receiver's variable) it determines the number of packets that TCP can send into the network. However, in practice, it is not easy to trace this variable without modifying the kernel code of the operating system. As a result, in order to measure (estimate) the congestion window, we measure the number of out-going packets in the network. These are the packets that are sent but not yet acknowledged. Another important metric of TCP is the round-trip time (RTT). To measure the round-trip time, we can use the PING utility. However, it should be mentioned that PING uses ICMP packets to estimate the round-trip time whereas in a TCP connection, we deal with data packets. In fact TCP uses Karn's algorithm, [KaPa87], to estimate the round-trip time and this is also a formal requirement for any TCP implementation, [RFC2988]. These are the basic metrics of TCP. However, as new theoretical approach requires *new kind of statistics and metrics*, it is essential to know how to measure these new metrics firstly for the sake of validation. Secondly, the new metrics themselves provide us new perspective and insight into the dynamics of TCP.

In this chapter, we introduce new metrics of TCP, design and implement a class of novel algorithms to measure those metrics. The algorithms include the measurement of the number of forward-going packets, the detection of the states of TCP and the measurement of state-based metrics of TCP. We also show the applicability of the new metrics and novel algorithms in TCP modelling and analysis.

The chapter is organized as follows. Section 2.1 discusses the background of TCP. In Section 2.2, a new virtual queue concept (metric) as well as a novel algorithm to measure the new metric is presented. A new algorithm for state detection of TCP is provided in Section 2.3. In addition, novel algorithms to collect the state-based metrics of TCP are also given in Section 2.3.

2.1 Background on TCP

2.1.1 Early TCP implementations

Modern TCP implementations contain a number of algorithms aimed at controlling network congestion while maintaining good user throughput and low delay. Early TCP implementations followed a *Stop-and-Wait* or a *Go-Back-N* (see [Sch97] for detailed operations and performance) model using cumulative positive acknowledgement and requiring retransmit timer expiration to re-send data lost during transport. These TCP implementations did little to minimize network congestion but contained basic elements of the transport protocol that today's TCP implementations inherit.

Stop-and-Wait protocol

In this procedure only one packet at a time can be transmitted. The transmitter then waits for an acknowledgment (ACK) or negative acknowledgment (NACK). Negative acknowledgments are used to indicate an error condition. If either a NACK arrives or a timeout expires, the packet is retransmitted. Only when an ACK arrives is the packet in question dropped from the transmit buffer. This is an appropriate protocol for halfduplex transmission, in which the two sides alternate transmission. However, this protocol obviously suffers from reduced throughput in the fullduplex case, particularly if the link propagation delay is significantly longer than the packet transmission time.

Go-Back-N protocol

In this procedure packets are transmitted continuously, if available, without waiting for an ACK. On receipt of a NACK, or expiration of the Time Out without receipt of an ACK/NACK, the packet in question and all packets following are transmitted. The pipelining of packets obviously improves the throughput performance. In practical versions of a *Go Back N* protocol, not all packets need be acknowledged. An ACK may positively acknowledge a given packet and all packets preceding it.

2.1.2 Modern TCP implementations

In this section, we overview some basic features of modern TCP implementations that we will deal with in this dissertion. During establishment or connect phase, the two ends exchange SYN packets for synchronization and generate initial sequence number. TCP provides orderly transfer of data packets during the subsequent data transfer phase. Finally, the protocol includes some procedures for terminating communication.

The TCP receiver: accepts packets out of sequence number order, buffers them in TCP buffer, and delivers them to its TCP user in sequence. Since the receiver has a finite sequencing buffer, it advertises a maximum window size, W_{max}, at connection setup time,

and the transmitter ensures that there is never more than this amount of unacknowledged data outstanding. The receiver returns an *acknowledgement (ACK)* for every good packet that it receives. The ACKs are *cumulative*, i.e., an ACK carrying sequence number n acknowledges all data up to, and including the sequence number $n-1$. The ACKs from the receiver carry the *next expected* packet number, which is the first among the packets required to complete the sequence of packets in the sequencing buffer. Thus, if a packet is lost, then the transmitter keep getting ACKs with the sequence number of the first packet lost repeatedly. We call these the *duplicate ACKs*.

The TCP transmitter: maintains the following variables for each connection (at any given time t):
$A(t)$ = the *lower window edge*; all the data numbered upto and including *A(t)*-1 has been transmitted and acknowledged. $A(t)$ is nondecreasing.
$W(t)$ = the *congestion window* (we also refer to this variable as *cwnd* in this dissertation). The transmitter can send packets with the sequence number n, where $A(t) \leq n < A(t) + W(t)$ and $W(t) \leq W_{max}$.
$W_{th}(t)$ = the *slow-start threshold* (we also refer to this variable as *ssthreshold* in this dissertation). $W_{th}(t)$ controls the increments in $W(t)$ as described later.

Tahoe TCP

The Tahoe TCP implementation, [Jac88], added a number of new algorithms and refinements to earlier implementations. These algorithms include : Slow Start, Congestion Avoidance, and Fast Retransmit. The refinements include a modification to the round trip time estimator used to set retransmission Time Out values. With Slow Start, when the congestion window smaller than threshold window, the connection load increases exponentially due to the fact that after every acknowledgment of a new packet, the sender increases its congestion window by one packet. When the network gets congested indicated by packet lost, the sender enters the Congestion Avoidance Phase. The threshold window is assigned half the value of the current congestion window. In this phase, the connection load increases linearly due to the fact that the sender increases its congestion window by $1/W$, where W is the current congestion window size, each time a new ACK arrives. With Fast Retransmit algorithm, after receiving a small number of duplicate acknowledgments for the same TCP packet, the transmitter assumes that a packet has been lost and retransmit the packet without waiting for the retransmission timer to expire, leading to higher channel utilization and connection throughput. In short :

1. After every acknowledgment of a new packet: if $W < W_{th}$ set $W = W + 1$: **Slow Start Phase** else set $W = W + 1/W$: **Congestion Avoidance Phase**.

2. After a packet loss is detected: set $W_{th} = W/2$; set W = 1:**Fast Retransmit Phase**.

Reno TCP

The Reno TCP implementation, [RFC2001], also used the enhancements incorporated into Tahoe, but modified the Fast Retransmit operation to include Fast Recovery. The new algorithm prevents the communication path from going empty after Fast Retransmit, thereby avoiding the need to Slow Start to refill it after a single packet loss. Fast Recovery is entered by a TCP sender after receiving an initial threshold of duplicate ACKs. This threshold is usually set to three. Once the threshold of duplicate ACKs is received, the sender retransmits one packet and reduces it window by one half. Instead of slowstart, as performed by Tahoe sender, the Reno sender uses additional incoming ACKs to clock subsequent outgoing packets. In short:

1. After every nonrepeated acknowledgment, the algorithm works as before: if $W < W_{th}$, set $W = W+1$: **Slow Start Phase** else set $W = W+1/W$: **Congestion Avoidance Phase**.

2. When the duplicate acknowledgments exceed a threshold, retransmit *next expected* packet; set $W_{th} = W/2$, then set $W = W_{th}$ and enter **Fast Recovery Phase**, resume congestion avoidance using new window once retransmission is acknowledged.

3. Upon timer expiration: **Time Out Phase**, the algorithm goes into **Slow Start Phase** as before : set $W_{th} = W/2$; set $W = 1$.

NewReno TCP

The NewReno TCP, [RFC2582], is a slightly modified version of Reno TCP. It includes a small change to the Reno algorithm at the sender that eliminates Reno's wait for a retransmit timer when multiple packets are lost from a window. The change concerns the sender's behaviour during Fast Recovery when a partial ACK is received that acknowledges some but not all of the packets that were outstanding at the start of the Fast Recovery Period. When multiple packets are lost from a window, NewReno can recover without a retransmission **Time Out**, retransmitting one lost packet per roundtrip time until all of the lost packets from that window have been retransmitted.

SACK TCP

The SACK TCP (Selective Acknowledgment TCP), [MMFR96], was introduced to improve the performance of TCP in presence of bursty losses. SACK TCP maintains a variable called *pipe* that represents the estimated number of packets outstanding in the path. The sender only sends new or retransmitted data when the estimated number of packets in the path is less than the congestion window. The variable pipe is incremented by one when the sender either sends a packet or retransmits an old packet. It is decremented by one when the sender receives a duplicate ACK packet with a SACK option reporting that new data has been received at the receiver. This is the reason why it is called *selective*. The sender

exits Fast Recovery when a recovery acknowledgment is received acknowledging all data that was outstanding when Fast Recovery was entered.

2.2 Dynamics of the virtual queue

2.2.1 The virtual queue concept

The congestion window of a TCP connection can be approximated by the number of outgoing packets in the network (i.e. the number of packets that are sent but not yet acknowledged). This number consists of:

- The number of packets flying towards the receiver (but not yet arrived).
- The number of packets stored at the receiver's buffer (but not yet processed).
- The number of acknowledgements flying back to the sender (but not yet received by the sender).

We are interested in the first metric (i.e. the number of packets flying to the receiver). These packets are either on propagating or are waiting at some queues of some intermediate routers. If we imagine all these queues as a single "virtual queue" then our algorithm measures the fluctuation of this *virtual queue* over time. The motivation for us to address this concept is because while the congestion window represents the traffic control at *end-point*, the virtual queue shows the impact of the congestion window on the *network*. In addition, it also clarifies the difference between traffic generated at end-points and traffic feeding the queues at the routers in a *feed-back* network, such as TCP networks. The idea is illustrated in Figure 2.1. A natural question arises then: How to measure the virtual

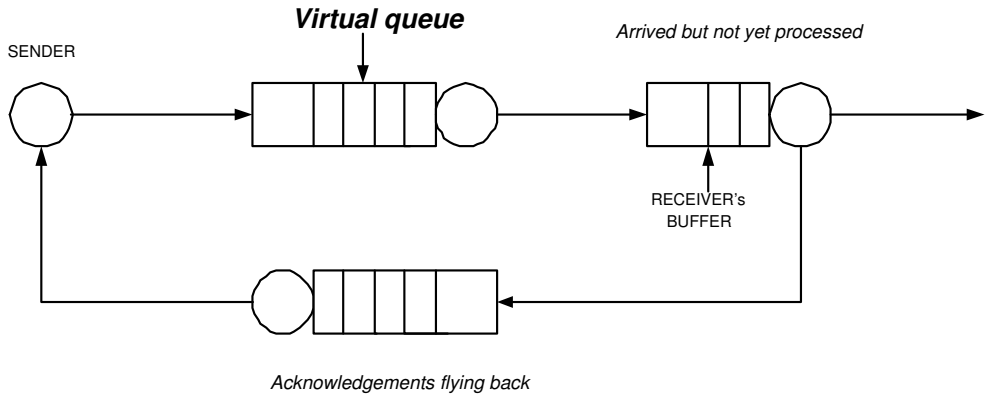


Figure 2.1: Illustration of the virtual queue

queue? To answer this question, we design a novel algorithm to measure this metric. The algorithm consists of the following steps:

1. At the beginning of the connection, the virtual queue is set to 0.
2. Synchronization of the sender and receiver time to a common time (or "absolute" time).
3. After synchronization, record the time stamp of the packets sent at the sender and received at the receiver.
4. Merge the time stamps recorded in increasing order.
5. Each time a packet is sent at the sender, the virtual queue is increased by 1 (a new packet enters the queue). Each time a packet is received at the receiver, the virtual queue is decreased by 1 (a packet leaves the queue).

One of the benefits of the proposed algorithm is that it works both for simulation traces as well as for measurement traces. As for simulation traces (obtained from the Network Simulator program (NS2), [NS2]), we did not need Step 1 because the simulator has already done the synchronization. However, for traces from real measurements (by using the TCPDUMP utility), synchronization was needed because the time at the receiver and the sender are not necessarily the same. In this case, we used the SYN packets in the TCPDUMP traces to synchronize the two clocks.

2.2.2 Results and validation

The virtual queue algorithm was verified using simulation and real measurements as well:

- The simulation study was carried out by using the NS2 tool. The algorithm was validated against different well-known versions of TCP (Tahoe, Reno, NewReno, SACK).
- Real measurements were carried out between the Budapest University of Technology and Economics (BUTE) and Ericsson Research at Budapest. The packets were recorded by using TCPDUMP tool. The algorithm was validated against the TCPDUMP traces obtained from the measurements.

An illustration of the dynamics of the virtual queue in Congestion Avoidance phase of TCP is shown in Figure 2.2. We also illustrate in Figure 2.2 the "induced delay" concept, which is the time elapsed between two consecutive bulks of data sent. This metric (delay) plays an important part in characterizing the forward delay and the backward delay of a TCP connection (note that the sum of the forward delay and the backward delay is the round-trip delay, i.e., RTT). In a symmetric network, these two kinds of delay are assumed to be equal. However, in an asymmetric network, they are not necessarily the same and the models have to take into account this effect.

In addition, the dynamics of the virtual queue also reveals to us the signal of congestion through the "congestion prone" line, the slope of which indicates congestion in the network. The more steeper the slope the more congested the network. It is because a number of packets are *permanently* queued in some intermediate routers and this number is increasing.

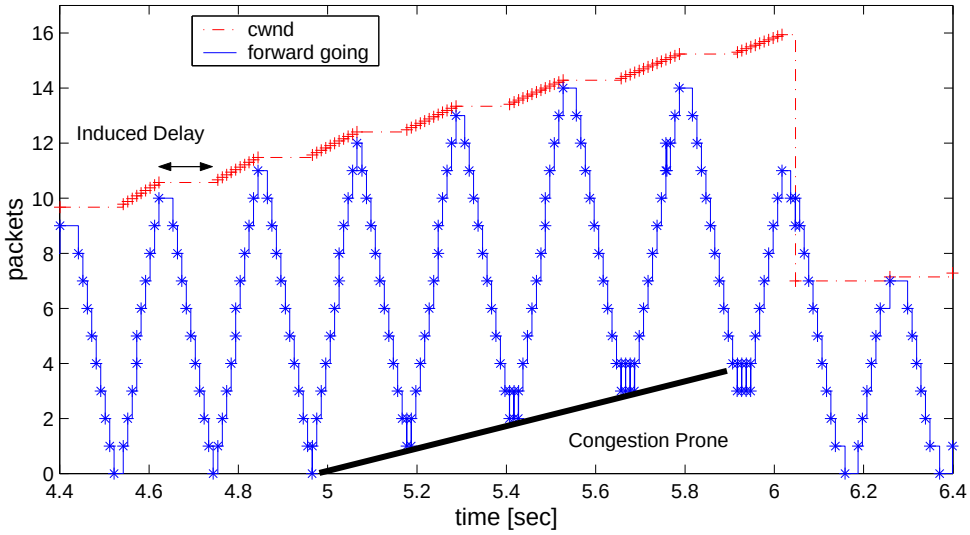


Figure 2.2: Dynamics of the virtual queue

The growing number of packets stored at the queues in the networks causes congestion in the network. When a queue is sufficiently large (generally it is the bottleneck link's queue), it starts dropping incoming packets. As illustrated in Figure 2.2, the TCP connection suffers a packet drop and halves its window in response to congestion in the network.

2.3 A State-based Analysis of TCP

As already mentioned previously, new theoretical approach requires *new kind of statistics and metrics*, it is essential to know how to measure these new metrics firstly for the sake of validation. Secondly, the new metrics themselves provide us new perspective and insight into the dynamics of TCP. In this section, we address some state-based issues of TCP. We try to give the answers to the following unanswered (or even unaddressed) questions:

- What is sojourn time distribution at each state of TCP during a connection?
- What is the jumping probability from one state to another state during a connection?
- What is the distribution of the number of packets sent in each round-trip time (RTT)?

We introduce a number of novel algorithms to measure these new metrics. At the core of the algorithms is the state detection scheme.

In what follows, we first describe the basic state detection mechanism then we discuss the difficulties involved with the implementation and our proposed solutions. Finally, we describe novel algorithms to answer the questions address above.

2.3.1 The state detection mechanism

To begin with, all TCP connections, after hand-shake phase, start with Slow Start phase to estimate the available bandwidth of the network. The TCP sender uses the congestion window variable (as well as the slow start threshold and some other variables) to control the number of packets sending to the network. The idea of our state detection algorithm is based on the dynamics of the congestion window and the slow start threshold process. With the congestion window, we can detect the *changes* of the states. Observe that if TCP is in some state and the congestion window is *increasing* then TCP *stays* in that state. If the congestion window is halved or decreased to 1, then a state change has happened. The slow start threshold provides us the details about the *next* state, if a state change is detected. An example of the operation of the state detection algorithm is illustrated in Figure 2.3. The

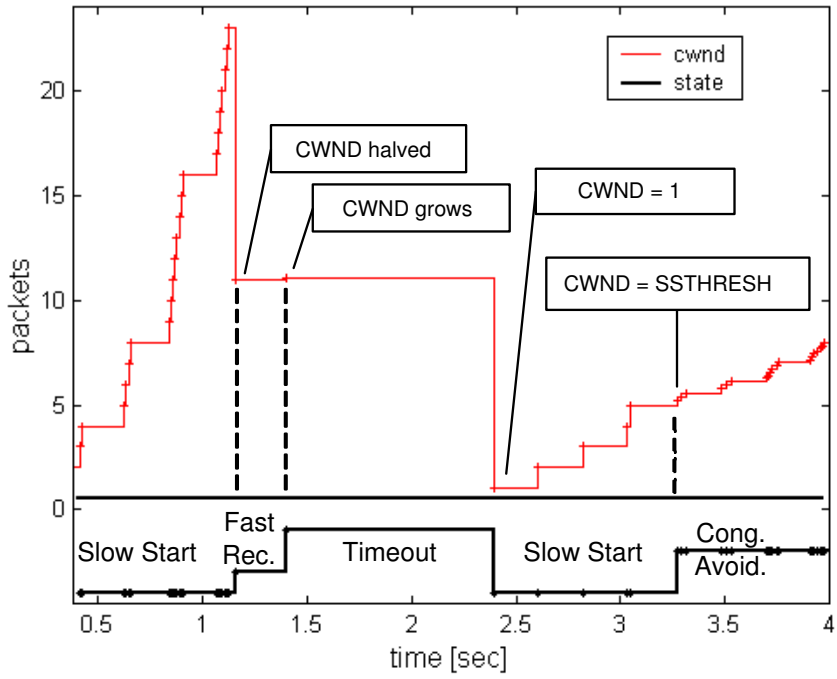


Figure 2.3: Illustration of the state detection algorithm

TCP connection starts with the Slow Start phase, so at the beginning the state variable is set to Slow Start. As far as the congestion window (*cwnd*) is *increasing*, TCP *stays* in Slow Start. The event that the congestion window is *halved* signals the end of the Slow Start phase and TCP (Reno) enters Fast Recovery. The arrival of the Recovery ACK signals the end of the Fast Recovery phase. If the *next* value of the *cwnd* is 1, then TCP Reno jumps from Fast Recovery to Time Out (otherwise it jumps to Congestion Avoidance). The event that the *cwnd* is set to 1 signals the end of the Time Out. After existing Time Out TCP

jumps to Slow Start phase. The event that the *cwnd* equals the the *ssthresh* signal the end of the Slow Start phase and TCP Reno jumps from Slow Start to Congestion Avoidance. The algorithm was implemented in a state detection tool. The flow diagram of the tool is

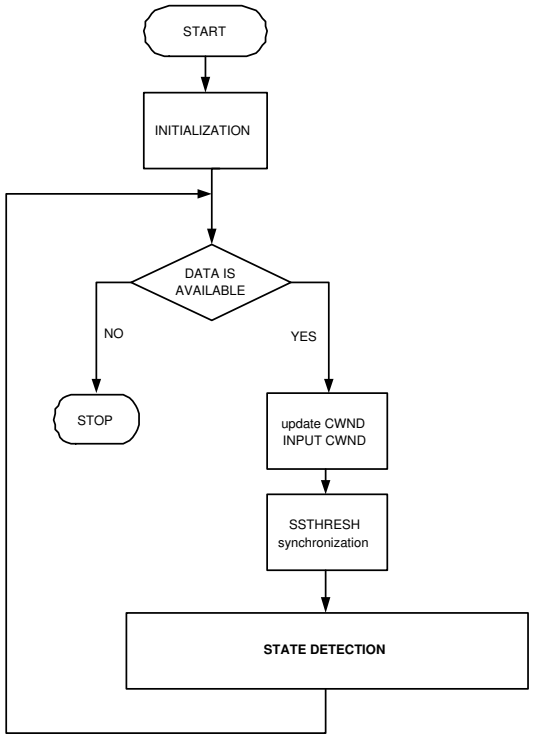


Figure 2.4: Flow diagram of the state detection tool

illustrated in Figure 2.4 (more details are provided in Appendix A.1). Now, let us discuss the difficulties we faced when implementing state detection.

2.3.2 Problems with state detection

We faced some difficulties when implementing the state detection mechanism of TCP. In this section, we first state the problems, then we discuss the solution for them.

Problem 1

The first difficulty is the detection of the end of Time Out when TCP *backs-off* more than one time in Time Out. As long as TCP is in Time Out, the congestion window is constantly 1 and each time it backs-off, the slow start threshold is halved. In this case, in order to check how many times TCP backs-off we need to introduce a new variable, namely *ssthreshControl* to follow the halving of the slow start threshold.

Problem 2

Another difficulty is the version of TCP that we deal with. The state detection of Reno TCP, NewReno TCP, SACK TCP are more or less the same. The situation is different with Tahoe TCP. In TCP Tahoe, there is *no* Fast Recovery. It *slow starts* after resending the lost packet(s), of any kind. In TCP Tahoe, we have two kinds of Slow Start: Slow Start after Time Out and Slow Start after triple ACKs. So in this case, we need to use the trace file that contains the information about the duplicate acknowledgements.

Problem 3

We observe from our simulations that when the congestion window is small (less than 4), the event that the congestion window is decreased to 1 does not necessarily means Time Out. Again, we need the trace file that contains the information about the duplicate acknowledgements to deal with this ambiguity.

We believe these are the major problems that we had to deal with. There are still many problems relating to the state detection mechanism that we have fixed but, for the sake of simplicity, are not listed here.

2.3.3 Measurement of state-based metrics of TCP

Being able to detect the beginning and the end of the states of TCP, it is now possible to design new algorithms to measure the state-related metrics of TCP. The metrics include:

- The sojourn time distribution at each state during a TCP connection.
- The jumping probabilities from one state to another state during a connection.
- The distribution of the number of packets sent in each time slot (RTT).

As the first step, the algorithms begin with the state detection phase.

1. The sojourn time distributions were computed by the following steps:
   - Collect all the possible states.
   - For each state, e.g. state i, measure frequencies of the time (in RTT) TCP spent at state i.
   - The set of frequencies constitutes the sojourn time distribution at each state.
2. The jumping probabilities from one state to another state are computed by the following steps:
   - Collect all the possible state jumps.
   - For each state, e.g. state i, count the total number of jumps from state i (N_i^{total}).
   - Count the the total number of jumps from state i to state j (N_i^j).

- $p_{ij} = \frac{N_i^j}{N_i^{total}}$.

3. The distributions of the number of packets sent at each time slot (RTT) at different states were computed by the following steps:
   - Collect all the possible states.
   - For each state, at each time slot, count the the total packets sent. Collect all the possible values.
   - Count the frequencies of each value.
   - The frequencies of the number of packets sent at each time slot constitutes the distributions.

The new state-based metrics provide us new insights into the dynamics of TCP. They are also indispensable in the state-based analysis which are discussed in details in the following sections. The applicability as well as the validation of the algorithms are presented in Chapter 3.

2.4 Conclusion

In this chapter, new concepts and novel algorithms to characterize the metrics of TCP were presented. We introduced the virtual queue concept (metric) that characterizes the dynamics of a TCP connection in forward direction. A novel algorithm to measure this metric was also provided and validated. In addition, we also introduced a novel algorithm to detect state changes of TCP during a connection. Being able to detect the state change, we designed and implemented new algorithms to collect the state-based metrics of TCP.

Chapter 3

A Unified Model for TCP

The modelling approaches found in the literature can be divided into two main groups: black-box modelling and white-box modelling. Black-box modelling approaches usually start from a theoretical model while white-box modelling approaches try to mimic inherent operations of TCP based on some statistics. An example of white-box modelling is the well-known ON/OFF model for voice traffic. It has two states: SILENCE and SPEAK. If the speaker speaks, then it is in the SPEAK state, and it is in the SILENCE state otherwise. With some probability the process jumps from SILENCE state to SPEAK state and respectively, with some probability the process jumps from SPEAK state to SILENCE state. So if the sojourn time distributions at the state are exponential (continuous time) or geometrical (discrete time), then the background process can be modelled by a two-state Markov chain and the traffic generated by voice sources can be well modelled by a Markov Modulated Poisson Process (MMPP), [HeLu86]. A natural question arises then: How about TCP traffic? Can we build a model for TCP by mimicking its inherent operations? The answer is yes, as we will illustrate later on in this chapter.

Another issue of TCP modelling is the versions of TCP in consideration because each TCP version has special mechanisms that should be carefully dealt with. Previous efforts [Kum98, CaMe00, ZCR00] to provide a unified model for different versions of TCP found in the literature differ from each others in one or another sense, but they are all in the *congestion window based modelling* paradigm. They all tried to study the dynamics of the *congestion window* (based on some Markovian assumptions) to estimate (model) the performance of TCP. We argue that the approach to model the dynamics of the congestion window by a Markov chain with the state space containing all the possible values of the congestion window would result in a huge number of states when the congestion window is getting sufficiently large. Recent developments of TCP, such as FAST TCP, HighSpeed TCP, Scalable TCP ([JWL04, Flo03, Kel03], respectively) that allow the congestion window as large as tens of thousands of packets would require a Markov chain as large as tens of thousands of states. Despite the fact that we do have a large body of literature on numerical methods for solving Markov chains (e.g. matrix-geometric method, see [Neu81] for a comprehensive review on the issue), the Markov chain of this magnitude is computationally infeasible in practice.

In contrast to the congestion window based approaches described above, in Chapter 3 we introduce a new perspective into the modelling paradigm of TCP. We study the dynamics of the *states* of the TCP itself in order to model it. We investigate the sojourn time distributions at the states as well as the jumping probabilities between the states and use the results achieved from the investigation to build a model to estimate TCP throughput. The main advantages of our approach are that (1) it provides a *unified* model for all well-known versions of TCP because they share the same logical set of states, and (2) the number of states is *significantly* reduced (in contrast with the number of possible values of the congestion window). Reducing the number of states in a Markov model makes it more computationally feasible and more analytically tractable, and thus, we believe, more desirable. Certainly, there is a trade-off between simplicity and accuracy and this issue is discussed in more detail in the following sections in this chapter. However, our guiding standpoint is to make the model *as simple as possible* and to increase the complexity of the model only if it is necessary.

In this chapter, a state-based modelling of TCP traffic is presented. During a connection, TCP stays in either of the following states: Slow Start, Congestion Avoidance, Loss Recovery (Fast Recovery and/or Fast Retransmit) and Time Out. We consider the states of a TCP connection as the *phases* of a stochastic process. We propose the use of the discrete-time batch Markovian arrival process (D-BMAP) to model the traffic generated by a TCP connection. The main contributions of this chapter are the followings. Firstly, we provide *a simple unified model* for some well-known versions of TCP based on the D-BMAP process. Secondly, we introduce a new concept, namely the *TCP characterization matrix* for a TCP connection that characterizes the transition probabilities between the states of TCP. This matrix is crucial in our state-based analysis. We also discuss the trade-offs between simplicity and accuracy in the state-based approach. Finally, we use simulation and numerical analysis to validate our proposed model.

The chapter is organized as follows. In Section 3.1 we present our state-based model for TCP. Numerical analysis of the proposed model is provided in Section 3.2. Results and validation of the model are presented in Section 3.3. Finally, Section 3.4 concludes the chapter.

3.1 A D-BMAP model for TCP stationary throughput

3.1.1 Background on D-BMAP

The D-BMAP process was originally introduced and examined in detail in [BloCa95]. The idea of D-BMAP can be traced back to M. Neuts' work in [Neu81]. Consider the discrete-time Markov chain with transition matrix $\mathbf{P}$. Suppose that at time k this chain is in some state i, $0 \leq i \leq N$. At the next time instant $k+1$, a transition to another or possible the same state takes place and a batch arrival may or may not occur. With probability $(p_0)_{i,j}$, $0 \leq i \leq N$, there is a transition to state j without an arrival, and with probability $(p_n)_{i,j}$,

$n \geq 1$, there is a transition to state j with a batch arrival of size n. We have that

$$\sum_{n=0}^{\infty}\sum_{j=0}^{N}(p_n)_{i,j} = 1 \tag{3.1}$$

Clearly the matrix $\mathbf{P}_0$ with elements $(p_0)_{i,j}$ governs transitions that correspond to no arrivals, while the matrices $\mathbf{P}_n$ with elements $(p_n)_{i,j}$, $n \geq 1$ govern transitions that correspond to arrivals of batches of size n.
Hence, the process can be defined as a two dimensional discrete-time Markov process $\{(N(k), J(k)), k \geq 0\}$ on the state space $\{(n, j), n \geq 0, 0 \leq j \leq N\}$ with the transition matrix

$$\mathbf{T} = \begin{pmatrix} \mathbf{P}_0 & \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3 & \dots \\ \mathbf{0} & \mathbf{P}_0 & \mathbf{P}_1 & \mathbf{P}_2 & \dots \\ 0 & 0 & \mathbf{P}_0 & \mathbf{P}_1 & \dots \\ \vdots & \vdots & \ddots & \ddots & \dots \\ \vdots & \vdots & \ddots & \ddots & \dots \end{pmatrix}$$

The variable $\{N(k), k \geq 0\}$ represents the counting variable and $\{J(k), k \geq 0\}$ the phase variable. With this notation the transition from state (l, i) to state $(l + n, j)$ correspond to an arrival of size n and a phase change of i to j. The matrix $\mathbf{P} = \sum_{n=0}^{\infty} \mathbf{P}_n$ is the transition matrix of the modulating Markov chain. The average *rate* ($\overline{BW}$) of the source characterized by the D-BMAP process mentioned above can be computed as follows:

$$\overline{BW} = \Pi \sum_{i=0}^{\infty} i\mathbf{P}_i\overline{e}$$

Furthermore, the stationary *distribution* can also be computed as follows

$$Pr[BW = i] = \Pi\mathbf{P}_i\overline{e}$$

3.1.2 The Model

The general case

In this section we will discuss how to apply the D-BMAP process to model the traffic generated by a TCP connection in a slightly different manner as in [BloCa95]. We propose a discrete-time model for TCP. The states of the background process (modulating process) are the states of TCP itself (i.e., Slow Start, Congestion Avoidance, Loss Recovery and Time Out).
Let us consider a generalized model of discrete MAP with batch arrivals:

- The process is time-slotted: the slot length is the average round-trip time ($\overline{RTT}$).

- The probability of transition from state i to state j is denoted by p_{ij} and the transition probability matrix of the modulating Markov-chain is $\mathbf{P} = \{p_{ij}\}$.
- When the chain is in state l, the TCP source transmits a random number of packets with probability generating function (p.g.f.) $B_l(z) = \sum_i b_i^{(l)} z^i$, where $b_i^{(l)}$ denotes the probability of i arrivals in a slot when the Markov chain is in state l.

Now, let us define $\mathbf{B}(z)$ matrix as follows:

$$\mathbf{B}(z) = \begin{pmatrix} p_{00}B_0(z) & p_{10}B_0(z) & ... & p_{N0}B_0(z) \\ p_{01}B_1(z) & p_{11}B_1(z) & ... & p_{N1}B_1(z) \\ . & . & ... & . \\ . & . & ... & . \\ p_{0N}B_N(z) & p_{1N}B_N(z) & ... & p_{NN}B_N(z) \end{pmatrix}$$

Let Π denote the stationary (limit) distribution of the modulating Markov chain. Then we can estimate the long term average throughput ($\overline{BW}$) of a TCP connection as follows:

$$\overline{BW} = \Pi(\mathbf{B}'(1))^T \overline{e}[MSS/\overline{RTT}]$$

where $\overline{e}$ is the unit column matrix defined by $\overline{e} = [1, 1, ..., 1]^T$ and $\mathbf{B}'(1) = d\mathbf{B}(z)/dz|_{z=1}$. It should be noted that the two interpretations described above are identical. The only difference is that we need to collect the statistics differently. In the first interpretation, we stick on the *states* first, then we consider the batch size. In the later interpretation, we first start with the *batch size*, then we examine all the possible state transitions corresponding to that batch size.

Now let us turn our attention to the computational (numerical) aspect of the above formula. First, let us determine Π. Since Π is defined as the stationary distribution of the modulating Markov chain ($\mathbf{P}$), it must satisfy the following equation: $\Pi = \Pi\mathbf{P}$. Consequently, to compute Π, we have to solve the linear systems of equations (with constraint that the sum of elements of Π is 1). This is a well-known problem and we can use any of the methods available in the literature. A little more subtle question is how to determine $\mathbf{B}'(1)$. Suppose that we already know the elements of $\mathbf{P}$, so what we still have to compute are $d\mathbf{B}_l(z)/dz|_{z=1}$, for $l = \overline{0, N}$. These values are actually the expectations of the distributions of the number of packets sent in a slot, state by state. We estimate these values by the average of the samples from simulations. Before turning to the next part, we would like to notice that even though we can estimate the *distributions* of the number of packets sent in one slot (state by state), what we really need is only the *expectations* of the distributions.

On the number of states of the model

Now we turn our attention to the problems related to the number of states the model should have. Especially, when we model Time Out (Exponential Back-off), the main question here

is whether one state is sufficient or not. This question yields to the validity of the Markovian assumption. For example, modelling Time Out with Exponential Back-off by only one state of the Markov chain is usually not sufficient because it is well-known that the sojourn time distribution in this case is not exponential (it is in fact heavy-tailed). But increasing the number of states increases the complexity of the model. So we have to find a compromise here. As long as there is only one RTO of Time Out, we model it by one state. Otherwise, Time Out is modelled by a Markov chain itself (with the number of states is the number of "Back-offs"). Figure 3.1 shows the Markov chain representation of the exponential back-off

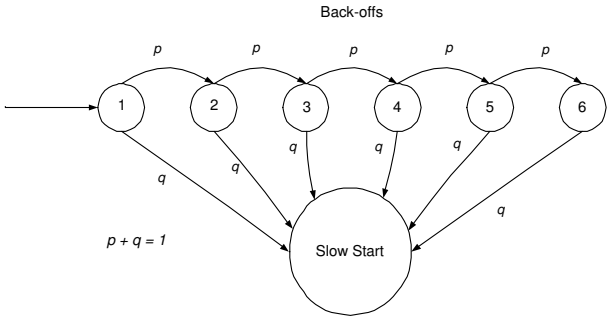


Figure 3.1: Markov chain representation of exponential back-offs

mechanism of TCP in Time Out period. Denote p the probability that TCP back-off and q is the probability that TCP leaves Time Out and enters Slow Start. Since when in Time Out, TCP either *backs off* or leaves Time Out, we have $p + q = 1$. We can estimate the value p and q from the statistics gained from simulations.

3.2 Numerical analysis

The main purpose of this section is to give a formula for stationary throughput of TCP in *closed form* with some *assumptions* to ease the analysis. In the following discussion, state 0 stands for Slow Start, state 1 stands for Loss Recovery, state 2 stands for Time Out and state 3 stands for Congestion Avoidance, respectively.

3.2.1 The TCP Reno case

In TCP Reno we assume that the duration of Loss Recovery is typically one RTT. It is because at the end of an RTT, the sender can decide to get out of Fast Recovery and continue in Congestion Avoidance or Time Out will occur. In other words, if TCP is in Fast Recovery then the probability of staying in Fast Recovery in the next round-trip time is assumed to be 0. We experience from most of our simulations that Time Out occurred (if any) only in one RTO and no Exponential Back-off. Although our general model can deal with Exponential Back-off, for the sake of simplicity, we deal mainly with

Time Out that lasts for only one RTO and as a consequence, TCP jumps to Slow Start with probability 1. Denote p_{TD} the probability of triple ACK loss event and p_{TO} the probability of Time Out event. Let $p_{loss} = p_{TD} + p_{TO}$. In this way, we have the probability that TCP jumps from Loss Recovery to Congestion Avoidance (p_{13}) is $\frac{p_{TD}}{p_{loss}}$ and the probability that TCP jumps from Loss Recovery to Time Out (p_{12}) is $\frac{p_{TD}}{p_{loss}}$. Now let us determine p_{01} and p_{31}. The fact that TCP jumps from Slow Start to Loss Recovery reveals to us that a loss has occurred and TCP was in Slow Start before the loss has been detected. As a result, we have p_{01}=P[loss occurred | from Slow Start]. Similarly, the event that TCP jumps from Congestion Avoidance to Loss Recovery implies that a loss has occurred and TCP was in Congestion Avoidance before the loss has been detected. Consequently, p_{31}=P[loss occurred| from Congestion Avoidance]. The probability of the event that TCP jumps directly from Slow Start to Congestion Avoidance (p_{01}) is P[cwnd_=ssthresh_]. Our simulation shows that, except for the first Slow Start, *no* packet is lost in Slow Start phase. This is understandable because Slow Start can only happen following a Time Out and Slow Start ends when the congestion window equals to the Slow Start threshold and TCP gets to Congestion Avoidance. After Time Out the pipe is already empty and the threshold value is sufficiently small so that it is easily reached by Slow Start phase and state change happens. That is why packet loss is very rarely detected in this period. Consequently, we assume that $p_{01} \approx 0$. From $p_{01} + p_{31} = p_{loss}$ we have $p_{31} \approx p_{loss}$ and consequently $p_{33} \approx (1 - p_{loss})$. Now denote $p_{threhold}$=P[cwnd_=ssthresh_] then we have $p_{03} = p_{threshold}$ and consequently $p_{00} = 1 - p_{threshold}$.

To sum up, the TCP characterization matrix for TCP Reno case can be filled as follows:

$$\mathbf{P_{Reno}} = \begin{pmatrix} (1 - p_{threshold}) & 0 & 0 & p_{threshold} \\ 0 & 0 & \frac{p_{TO}}{p_{loss}} & \frac{p_{TD}}{p_{loss}} \\ 1 & 0 & 0 & 0 \\ 0 & p_{loss} & 0 & 1 - p_{loss} \end{pmatrix}$$

where $p_{loss} = p_{TD} + p_{TO}$.

Let Π be the stationary distribution of the modulating Markov chain, $\Pi = (\pi_0, \pi_1, \pi_2, \pi_3)$. We have $\Pi = \Pi\mathbf{P}$ and Π is a distribution vector, so the following equation system holds:

$$\begin{aligned} \pi_0 &= (1 - p_{threshold})\pi_0 + \pi_2 \\ \pi_1 &= \pi_3 p_{loss} \\ \pi_2 &= \pi_1 \frac{p_{TO}}{p_{TD}} \\ \pi_3 &= \pi_0 p_{threshold} + \pi_1 \frac{p_{TD}}{p_{loss}} + \pi_3(1 - p_{loss}) \end{aligned}$$

with the constraint $\pi_0 + \pi_1 + \pi_2 + \pi_3 = 1$.

Algebraic computation yields:

$$\begin{aligned}
\pi_0 &= \frac{p_{TO}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}} \\
\pi_1 &= \frac{p_{loss}p_{threshold}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}} \\
\pi_2 &= \frac{p_{TO}p_{threshold}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}} \\
\pi_3 &= \frac{p_{threshold}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}}
\end{aligned}$$

Finally, we deal with the case when the slot times at the states are different. Let T_i be the slot time at state i, then we have the *corrected* stationary distribution $\alpha = (\alpha_0, \alpha_1, ..., \alpha_N)$ with $\alpha_i = \frac{\pi_i T_i}{\sum_j \pi_j T_j}$. Specifically, the time slot in Slow Start, Congestion Avoidance and Fast Recovery is roughly $\overline{RTT}$ whereas the time slot in Time Out (Exponential Back-off) is measured by $\overline{RTO}$. Denote $k = \frac{\overline{RTO}}{\overline{RTT}}$. Notice that in practice k is approximately equal to 4 ($k \approx 4$). Denote $\rho = \frac{1}{1+(k-1)\pi_2}$ be the multiplicative correction term. We have the *corrected* stationary distribution of the modulating Markov-chain can be expressed in *closed form* as follows:

$$\begin{aligned}
\pi_0 &= \frac{\rho p_{TO}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}} \\
\pi_1 &= \frac{\rho p_{loss}p_{threshold}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}} \\
\pi_2 &= \frac{k\rho p_{TO}p_{threshold}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}} \\
\pi_3 &= \frac{\rho p_{threshold}}{(1+p_{TD}+2p_{TO})p_{threshold}+p_{TO}}
\end{aligned}$$

Notice that if $k > 1$, then $\rho < 1$ and $k\rho > 1$. The fact that $\rho < 1$ implies that the *corrected* fraction of time that TCP stays in Slow Start, Congestion Avoidance and Loss Recovery is *smaller* than before correction. Similarly, the fact that $k\rho > 1$ implies that the *corrected* fraction of time that TCP stays in Time Out (Exponential Back-off) is *longer* than before correction. Since in Time Out events significantly reduce performance, without correction we might have *overestimated* the performance that TCP does actually produce.
Finally, the distributions (as well as the expected values) of the number of packets sent in each time slot for every state are estimated by simulations.

3.2.2 The TCP Tahoe case

As in the TCP Reno case, in TCP Tahoe, all four states are possible. However, the determination of the characterization matrix is different in a number of ways. Firstly, the Loss Recovery phase contains only the Fast Retransmit mechanism, without Fast Recovery (the

time spent in Loss Recovery is, nevertheless, the same as in TCP Reno case, i.e. approximately one RTT). Secondly, and most importantly, after Loss Recovery, instead of going to Congestion Avoidance phase, TCP Tahoe continues with Slow Start. So applying the same reasoning as in the TCP Reno case, the TCP characterization matrix for TCP Tahoe case can be filled as follows:

$$\mathbf{P_{Tahoe}} = \begin{pmatrix} (1 - p_{threshold}) & 0 & 0 & p_{threshold} \\ \frac{p_{TD}}{p_{loss}} & 0 & \frac{p_{TO}}{p_{loss}} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & p_{loss} & 0 & 1 - p_{loss} \end{pmatrix}$$

where $p_{loss} = p_{TD} + p_{TO}$.

3.2.3 The TCP New Reno and TCP SACK case

As we have mentioned in the previous section, the modelling of Loss Recovery phase of TCP New Reno and TCP SACK is not negligible since TCP, in order to avoid Time Out, stays relatively large amount of time in Loss Recovery phase to recover when multiple (consecutive or closely placed) packet losses occur within a window of data in transition. At this point, it seems that we might need a much more complicated characterization matrix to model the performance of TCP New Reno and TCP SACK. Interestingly, by contrast, it is not the case. TCP New Reno and SACK, instead of going into Time Out, compensate packet losses in Loss Recovery. Consequently, there is *no* Time Out phase in TCP New Reno and SACK, in most cases. We state "in most cases" because there is an exception when the congestion window is very small (to the extreme case when too many TCP flows competing (sharing) for a very slow bottleneck link). In this case, Time Out could happen even with TCP New Reno and TCP SACK because there is not enough acknowledgment for triple ACK case and TCP is forced to Time Out. We shall deal with this problem in a separated part, but for the time being, we assume that there is enough acknowledgements flying back and so there is no Time Out with TCP New Reno and SACK. The exclusion of Time Out implies that, apart from the *first* Slow Start, there is *no* Slow Start phase because Slow Start period could *only* happen after a Time Out period. Consequently, we have only two states in consideration, namely Congestion Avoidance and Loss Recovery. Notice that our simulations and measurements as well as results from earlier publications (e.g., [PFTK98]) suggested that the number of packets lost in a window can be approximated by a geometrical distribution. This implies that the time TCP Reno and SACK stay in Loss Recovery is geometrically distributed (Markov property). Furthermore, the probability of Time Out is approximately zero ($p_{TO} \approx 0$). As a result, the probability that TCP jumps from Congestion Avoidance phase to Loss Recovery phase is the probability of loss event (p_{loss}) and the probability that TCP stays in Congestion Avoidance is $1 - p_{loss}$. Let $p_{recovery}$ be the probability that TCP stays in Loss Recovery phase and consequently

$(1 - p_{recovery})$ is the probability TCP jumps from Loss Recovery to Congestion Avoidance. So the characterization matrix can be filled as follows:

$$\mathbf{P_{NewReno/SACK}} = \begin{pmatrix} 1 - p_{loss} & p_{loss} \\ p_{recovery} & 1 - p_{recovery} \end{pmatrix}$$

The stationary distribution (π_0, π_1) of this two-state Markov chain is thus $(\frac{p_{recovery}}{p_{loss}+p_{recovery}}, \frac{p_{loss}}{p_{loss}+p_{recovery}})$, respectively. At this point, it seems that the model of TCP is as simple as the well-known ON/OFF model for voice traffic. In a sense, it's true. However, it should be noticed that our model is different from the traditional ON/OFF model in a number of ways. Firstly, TCP, in our case, doesn't have OFF period. Data is sent actively in both states (Congestion Avoidance and Loss Recovery). Secondly, we assume *general* distributions for packets sent at each time slot in contrast to the very limited assumption of Poisson process as in the voice model. The determination of the distributions of the number of packets sent in each time slot (i.e. each round-trip time) in Congestion Avoidance and Loss Recovery is identical to the TCP Reno case.

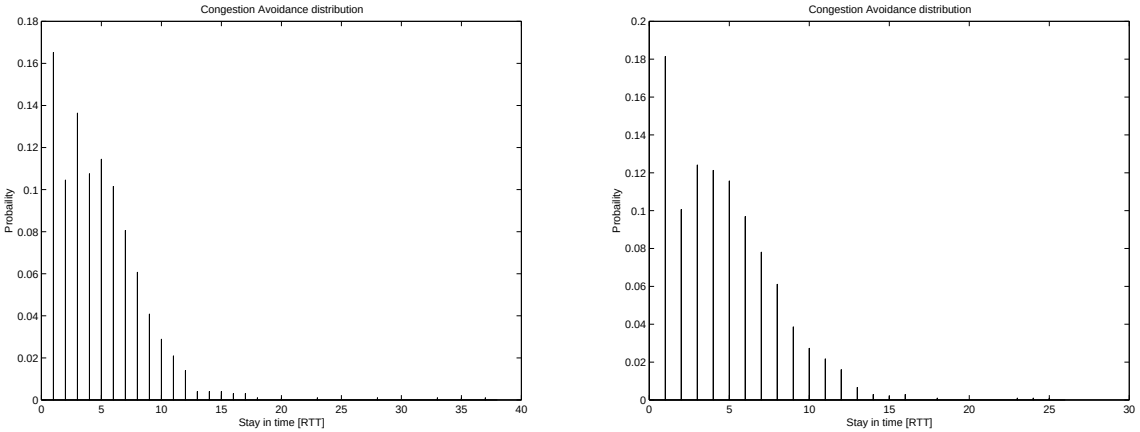
3.3 Results and Validation

We need to validate two things. Firstly, we consider the validity of the Markov property in our model by examining the sojourn time distribution at different states of TCP. Secondly, we use our model to estimate the steady-state throughput of a TCP connection (with different versions) by validating it against different packet loss probabilities. In order to do so, we first need to carry out experiments to collect the statistics mentioned in the model construction part. The experiments were conducted using ns-2 simulator. The topology that we used was a simple half-dumbbell topology. We added a loss module at the output port of the bottleneck link's router that can deliberately drop packets so that we can control the drop probabilities. In this way, instead of adding more connections to the background traffic we examine a *single* TCP connection. We believe that by deliberately tuning the loss probability, we are able to *emulate* different scenarios of background traffic because the effect of background traffic on a certain TCP connection ultimately results in the packet loss probability of that connection. For the details about our simulations, the packet size was 1000 bytes, the access link was 8 Mb/s with a delay of 0.1 ms, the bottleneck link was 800 Kb/s with a delay of 100 ms and the buffer size was 20 packets. Regarding the queue management schemes at the router, we consider both Drop-Tail and RED mechanisms. With RED-style queue managements, the *gentle* RED was used and the *adaptive* parameter was tuned in.

3.3.1 On the sojourn time distribution at the states

Congestion Avoidance

First, we examine the sojourn time distributions at Congestion Avoidance state. Since all versions of TCP perform identically in this state, we concentrate on Reno version. However, we examine Reno both with Drop Tail and RED router. As we can see in Figure 3.2, the sojourn time distribution at Congestion Avoidance is geometrically shaped both in Drop Tail and RED case. It supports the Markovian assumption of our model for this state.

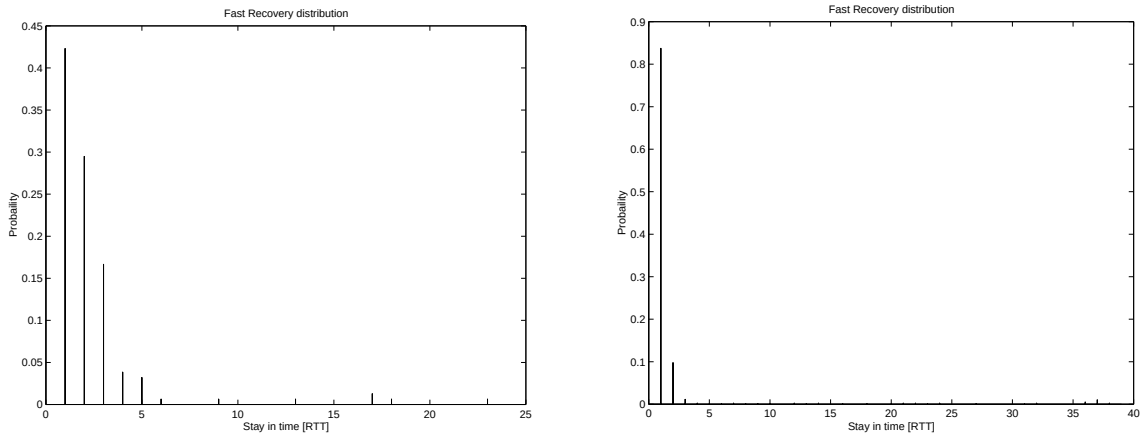


(a) (b)

Figure 3.2: Sojourn time distribution at Congestion Avoidance state: (a) Drop Tail case; (b) RED case.

Loss Recovery

In Reno, according to specification, it takes approximately one RTT for TCP to get out of Fast Recovery and TCP enters Congestion Avoidance or Time Out triggers. The situation is different with NewReno and SACK. These versions of TCP is equipped with mechanisms to avoid Time Out in case of multiple losses in a window by *longer* Fast Recovery. With NewReno and SACK versions TCP can stay in Loss Recovery for several round-trip times, depending on the number of losses occurred in a window. So here, we can talk about the distribution of sojourn time. As we can see in Figure 3.3, the sojourn time distribution at Congestion Avoidance is geometrically shaped both in NewReno TCP and SACK TCP cases. This confirms the Markovian behaviour of TCP in this state.



(a) (b)

Figure 3.3: Sojourn time distribution at Loss Recovery state: (a) NewReno TCP case; (b) SACK TCP case.

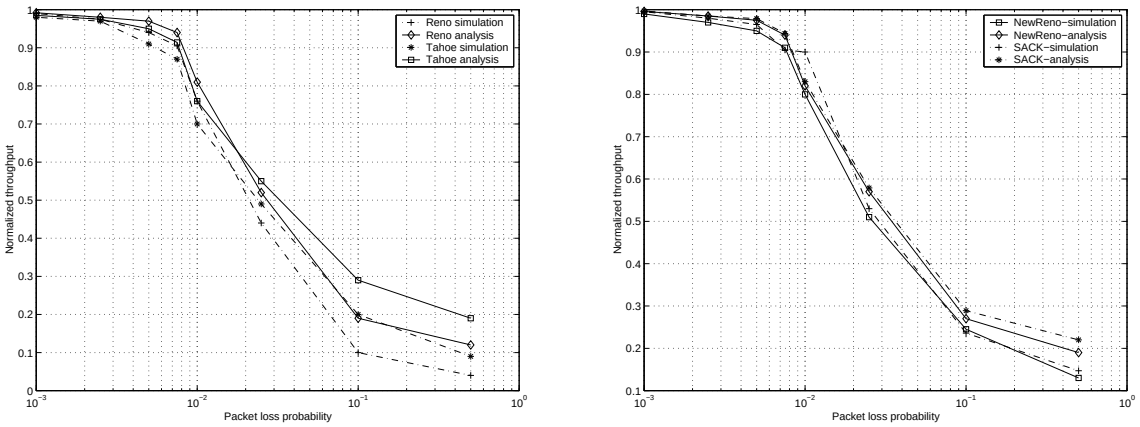
3.3.2 On the stationary performance of TCP

The stationary performance is one of the most important metrics of TCP. This section provides the validation of the stationary throughput of TCP. We compare our numerical results that we achieved from our analysis with the simulation results of NS2 under the same configuration. We go through versions of TCP, version by version.

The TCP Reno and Tahoe case

In TCP Reno and Tahoe case, all four states are possible. The probability that Time Out (Exponential Back-off) exists depend on the magnitude of the packet loss probability. If the loss probability is very small (less than 1 percent), then Time Out is rare with TCP Reno, at least with our configuration. If the loss probability is increased, then the probability of more packets dropped in a window of packets increases. Consequently, the probability of Time Out events also increases. We observe from our simulations that if the packet loss probability gets to 10 percent or higher, Time Out is frequent with Reno. This has severe effect on the performance of TCP Reno. So we validate our model in different packet loss scenarios. We basically examine three types of losses: small (less than 1 percent), average (up to 5 percent), high (higher than 10 percent).

Figure 3.4(a) shows the throughput of Reno TCP by simulation and analysis. It presents that our model is in accordance with the simulation results, although we experience some overestimation. However, the overestimation is small enough (less than 1 percent), especially when the packet loss probability is small. The overestimation is already discussed in previous Sections (assumption of exponentially distributed sojourn time as well as the presence of Exponential Back-off where we have given up some details for the sake of simplicity of our model). Regarding the relative performance of TCP Reno and Tahoe, we observe that when



(a) (b)

Figure 3.4: Throughput of TCP against different loss probabilities: (a)Reno-Tahoe TCP case; (b) NewReno-SACK TCP case.

packet loss probabilities are small, TCP Reno performs slightly better than TCP Tahoe *both in simulation and analysis*. As the packet loss probability gets higher, the situation changes. TCP Tahoe seems to perform better than TCP Reno, at least in our experiments. This is the reason why in wireless environment, when packet loss probability is high and not necessarily because of congestion, TCP Tahoe performs somewhat better than TCP Reno, as widely suggested in the literature.

The TCP NewReno and SACK case

In TCP NewReno and SACK case, we basically have only two states, namely Congestion Avoidance and Loss Recovery. TCP NewReno and SACK perform more or less identically most of the time. The only difference is in the Loss Recovery phase where TCP SACK, by adapting to the *pipe*, is a little bit more aggressive than TCP NewReno. This results in the unfairness between TCP NewReno and TCP SACK when they are in presence. Figure 3.4(b) shows the throughput of NewReno and SACK TCP by simulation and analysis. We observe that SACK performs slightly better than NewReno in most of the cases. This observation supports our view on the unfairness between TCP NewReno and TCP SACK. And surprisingly enough, the model error is *smaller* than in the Reno/Tahoe case. We believe that this is because of the existence of *Time Out* periods in these cases (i.e. Time Out greatly reduces Reno/Tahoe performance but NewReno and SACK are armed with mechanisms to deal with multiple losses in a window, thus avoiding Time Out). At this point we believe that Exponential Back-off is the major cause of error, but more analysis is still needed.

3.4 Conclusion

In this chapter, a unified model for some well-known versions of TCP based on the states of TCP itself was presented. We introduced a new concept, namely the TCP characterization matrix. We derived the relationship between the elements of the TCP characterization matrix and other metrics of the TCP/IP network. We showed how to use this matrix to model the stationary performance of TCP. The model was validated against different network scenarios, with different versions of TCP.

Chapter 4

A Game-Theoretic Analysis of TCP

An important issue regarding TCP is the impact of its rate control and parameter setting on the network. Basically, TCP is a closed loop control scheme. Congestion in the network is fed back to the source in the form of losses (Reno-like versions) or delay (such as TCP Vegas). The source then reacts to the congestion signal from the network by reducing its transmitting rate. In other words, we can consider packet loss and high queueing delay as the *cost* of (aggressively) sending packets into the network. The higher the rate, the higher the cost (certainly, the relationship is not necessarily linear in nature), given a fix network. Furthermore, as the Internet has been gradually transforming from a government sponsored project to a private enterprize (or even a commodity), the economics of the Internet becomes more and more important issue. Consequently, Internet connectivity and services will have to confront issues of pricing and cost recovery. In this perspective, the cost of congestion can be in monetary form. Introducing cost of congestion into the network creates balance, stability and high utilization of resource usage. The cost of congestion, in our case, can be either price or delay (the application is delay-sensitive). It is suggested in [MaVa94] that congestion pricing could be implemented by using "smart market" where price for sending a packet varies on a very short time scale. Specifically, for the TCP implementation, the congestion price is updated every round-trip time. Given the pricing schemes (parameter setting), a natural question arises then: Are these schemes efficient? Is there any equilibrium state from where no one has the incentive to deviate? Game theory (see [OsRu94] for a comprehensive introduction) seems to be the promising tool to answer this question. Schenker in his pioneering paper [Sch95] used game-theoretic approach to analyze the flow control mechanisms (for Poisson arrivals) with different queueing disciplines at the routers. Korilis *et al* in [Kor95] also used game-theoretic approach to study the existence of equilibria in noncooperative optimal flow control, especially those with QoS constraints. The analysis presented in these papers (and some others, e.g. [AlBa04]) deals with *general* flow control, ignoring some important inherent TCP operations. Regarding parameter setting of TCP, recently, Akella *et al* in [Ake02] also used the tools from game

theory to examine the behaviour of TCP Reno-like (loss-based) flow controls under selfish parameter setting. However, a comprehensive game-theoretic analysis of *delay-based* TCP (such as TCP Vegas) is still unavailable. TCP Vegas is of our particular interest because, firstly, it has *inherent pricing schemes* in its design that resemble the congestion pricing schemes proposed in the literature. We believe that by better understanding TCP Vegas' inherent pricing schemes, we will have a better insight into understanding and designing pricing schemes for TCP traffic in general. Secondly, with the emergence of very large bandwidth-delay product networks such as the transatlantic link with a capacity in the range of 1 Gbps - 10 Gbps, new transport protocols have been proposed to better utilize the network in these circumstances. One promising proposal is the FAST TCP [JWL04]. Since the design of FAST TCP is heavily based on the design of TCP Vegas, there is a need to reconsider the benefits as well as the drawbacks of TCP Vegas in order to have an insight into the performance and possible deployment of FAST TCP in the future Internet.

In this chapter, we use the tools from game theory to understand the impacts of the *inherent congestion pricing schemes* in TCP Vegas as well as the problems of parameter setting of TCP Vegas on its performance. It is shown how these inherent pricing schemes result in a rate control equilibrium state that is a Nash equilibrium which is also a global optimum of the all-Vegas networks. On the other hand, if the TCP Vegas' users are assumed to be selfish in terms of setting their desired number of backlogged packets in the buffers along their paths, then the network as a whole, in certain circumstances, would operate *very inefficiently*. We then extend our analysis to investigate FAST TCP, a recently proposed TCP Vegas-based protocol. Our analysis points out that FAST TCP, like Vegas TCP, benefits from its inherently efficient pricing schemes, but the parameter setting of FAST TCP is very vulnerable to selfish actions of the users. This poses a serious threat to the possible deployment of FAST TCP in the future Internet.

This chapter is organized as follows. The background on TCP Vegas is provided in Section 4.1. The TCP Vegas games are described and analyzed in detail in Section 4.2. Finally, Section 4.3 concludes the chapter.

4.1 Background

4.1.1 TCP Vegas

TCP Vegas was first introduced by Brakmo *et al* in [BMP94]. Basically, it is a delay-based congestion control scheme that uses both queueing delay and packet loss as congestion signal. TCP Vegas tries to control the number of packets buffered along the path with the targeted number to be between α and β ($\alpha \leq \beta$). Let $w(t)$ denote the congestion window at time t, RTT denote the round-trip time and $baseRTT$ is the smallest value of the round-trip time so far (actually, this is an *estimate* of the propagation delay). Denote $\mathsf{diff} = \frac{RTT - baseRTT}{RTT} w$, then the dynamics of the congestion window of TCP Vegas can be

expressed as follows:

$$w(t+1) = \begin{cases} w(t)+1 & \text{if } \mathsf{diff} < \alpha, \\ w(t)-1 & \text{if } \mathsf{diff} > \beta, \\ w(t) & \text{otherwise.} \end{cases} \tag{4.1}$$

In a TCP Vegas/REM network [ALLY01], a slight modification is introduced into the updating mechanism of the congestion window. Each link l (with capacity c_l) update the *link price* $p_l(t)$ in period t based on the aggregate input rate $x^l(t)$ and the buffer occupancy $b_l(t)$ as follows:

$$p_l(t+1) = [p_l(t) + \gamma(\mu_l b_l(t) + x^l(t) - c_l)]^+ \tag{4.2}$$

where $0 < \gamma$ and $0 < \mu_l < 1$ are scaling factors of REM. Each source will estimate the *total* price along its path and update its sending rate accordingly. To feed back the prices to sources, link l marks each arriving packet in period t, that is not already marked at an upcoming stream, with probability $m_l(t)$ defined as:

$$m_l(t) = 1 - \varphi^{-p_l(t)}$$

where $\varphi > 0$ is a constant. Once a packet is marked, its mark is carried to the destination and then conveyed back to the source via acknowledgement, like ECN scheme. The source i estimates the end-to-end marking probability by the fraction $\hat{m}_i(t)$ of its packets marked in period t, and estimates the path price $p_i(t)$ by:

$$\hat{p}_i(t) = -\log_\varphi(1 - \hat{m}_i(t))$$

The dynamics of the congestion window of TCP Vegas/REM can be expressed as follows:

$$w_i(t+1) = \begin{cases} w_i(t)+1 & \text{if } -\frac{w_i(t)}{RTT_i(t)} < \frac{\alpha}{\hat{p}_i(t)}, \\ w_i(t)-1 & \text{if } -\frac{w_i(t)}{RTT_i(t)} > \frac{\alpha}{\hat{p}_i(t)}, \\ w_i(t) & \text{otherwise.} \end{cases} \tag{4.3}$$

4.1.2 Goodput models of TCP Vegas

Throughout the chapter our game-theoretic analysis uses the models that are previously derived. These are:

Model 1: (Thomas Bonald's model)

In [Bon98], the goodput of multiple flows sharing a bottleneck link is analyzed (by using fluid approximation) both for TCP Reno and TCP Vegas case. Assume N TCP flows sharing a bottleneck link with capacity μ, propagation delay τ and buffer size B. The parameters of TCP are: α and β. The main results in their paper that we use in our analysis are the following:

- If $N\alpha < B$, there exists a finite time from which no loss occurs. In addition, the window size stabilizes in finite time. If $\alpha \neq \beta$, the congestion windows converge not to a single point (but a region). This implies unfairness among flows even in equilibrium. If $\alpha = \beta$, then $w_1 = w_2 = ... = w_N = \frac{\mu\tau}{N} + \alpha$ and the average rate $\lambda_1 = \lambda_2 = ... = \lambda_N = \frac{\mu}{N}$. Note that in this case the link is fully utilized.
- If $N\alpha \geq B$, then TCP Vegas behaves exactly like TCP Reno. Let $\omega = \frac{\mu\tau}{B}$ and γ is the multiplicative decrease of TCP Reno (typically $\frac{1}{2}$). If $\omega \geq \frac{\gamma}{1-\gamma}$ then $\lambda_{total} = \frac{(1-\gamma^2)(\omega+1)^2}{2(1-\gamma)(\omega^2+\omega)+1}\mu < \mu$. This implies that in this case the link is not fully utilized.

Model 2: (Steven Low's model)
Steven Low *et al* in [LoLa99, LPW02, Low03, ALLY01] described an optimization framework to study the performance of the TCP Vegas in a general network topology and under different queue management schemes at the routers. We would mention the result regarding the goodput of TCP Vegas under REM queue management scheme (TCP Vegas/REM) that we will use in our analysis later in this paper. It is proved in [LPW02] that the equilibrium rate of TCP Vegas can be calculated as: $\lambda_i = \frac{\alpha_i}{p^*}$, where p^* denotes the equilibrium price. Note that this result is true for a general network topology (not restricted to a single bottleneck link).

4.2 The TCP Vegas games

In this Section, the games regarding the inherent pricing schemes (for rate allocation) and parameter setting of TCP Vegas are described and analyzed in detail. We also investigate the impact of the results on the performance of TCP Vegas and on the network as a whole.

4.2.1 Game 1: Rate allocation of TCP Vegas

We consider a network that consists of a set $\mathcal{L} = \{1, 2, ..., L\}$ of links with capacity c_l, $l \in \mathcal{L}$. Assume that the network is shared by a set of flows (sources). The set of flows is denoted by $\mathcal{N} = \{1, 2, ..., N\}$. The rate of flow i is denoted by x_i, $i \in \mathcal{N}$. Flow i uses a subset ($\mathcal{L}_i$) of $\mathcal{L}$ in its path ($\mathcal{L}_i \subseteq \mathcal{L}$). Let us define the routing matrix as follows:

$$\mathbf{R}_{li} = \begin{cases} 1 & \text{if } l \in \mathcal{L}_i, \\ 0 & \text{otherwise.} \end{cases}$$

The physical capacity constraints of the flows therefore can be defined as :

$$\mathbf{Rx} \leq \mathbf{c} \tag{4.4}$$

where $\mathbf{x} = (x_1, x_2, ..., x_N)$ is the flow rate vector and $\mathbf{c} = (c_1, c_2, ..., c_L)$ is the link capacity vector. In addition, flow rates cannot be negative:

$$x_i \geq 0, \qquad i = 1, 2, ..., N \tag{4.5}$$

The set of flow rate vectors Λ that satisfy both conditions 4.4 and 4.5 is called a feasible set.

It should be mentioned that our network, as TCP network in general, assumes feedback-based flow control. The feedback can be implicit (e.g. queueing delay) or explicit (by pricing and/or using Explicit Congestion Notification (ECN)). The sources (end-points) use Vegas-style flow control, as defined in [BMP94], [LPW02]. We consider the flows as the players of the game. The strategy space for a player is the range of it sending rate.

Let us define the following generic payoff function for each player:

$$B_i(x_i) = \alpha_i \log(x_i) - \sum_{l \in \mathcal{L}_i} \int_0^{x_i} \pi_l(y) dy \tag{4.6}$$

and $\pi_l = p_l(\sum_{l \in \mathcal{L}_k} x_k)$ is defined as the function of the total flow rates on link l. This function is actually the *price* that is fed back to the player i sending at rate x_i, which is an increasing function. The higher the rate, the higher the price. Hence, the second term in Equation 4.6 can be interpreted as the bandwidth *cost* fed back to player i when it attempts to transmit at rate x_i. The first term in Equation 4.6 reflects the *gain* of player i when transmitting at rate x_i. This form is motivated by the results of Theorem 4 in [Low03]. Notice that this is a concave function of x_i, so it is also justified in practice since concavity implies diminishing return, a property that most models in practice should have. As a result, the payoff $B_i(x_i)$ represents the *net* benefit of player i when transmitting at rate x_i. The price (cost) can be communicated to the end-user (the player) by the mean of the total queueing delay of its packets in the path, as in TCP Vegas/Drop-Tail network. The price can also be communicated explicitly to the user by using REM active queue management scheme (with ECN) and here we have a Vegas/REM network. In the first case, we would like to mention that we *implicitly* use the PASTA property for Poisson arrivals with FIFO scheduling principle to derive the proportional relationship between the total rate arrive at the link and the queuing delay. We can also suggest here the Little's formula for this relationship. This is assumed frequently in the literature with or without any mention. In any case, if the aggregate arrival flow is not Poisson (e.g. self-similar traffic), then queue length (queueing delay) is generally larger than the Poisson one. Furthermore, the expression of queueing delay in our model is assumed to be *additive* among links. This is true for a Norton network with Poisson arrivals. So, strictly speaking, our analysis can be considered as a *worst case* analysis for TCP Vegas/Drop-Tail network. For TCP Vegas/REM network, the additive assumption is justified when the mark rates are small. Indeed, let $\pi_l(t)$ be the marking probability at link l at time t and the end-to-end marking probability $q_i(t)$ that the end-point i observes (and to which source algorithm reacts). For small $\pi_l(t)$, $q_i(t) = 1 - \prod_{l \in \mathcal{L}_i}(1 - \pi_l(t)) \approx \sum_{l \in \mathcal{L}_i} \pi_l(t)$.

Under the assumptions mentioned above, our problem can be modelled as a non-cooperative game. The strategy space for a player is its sending rate and is determined by the capacity of the links. The strategy for player i can be defined as $S^{(i)} = \{x_i | 0 <$

$x_i \leq c^i_{max}\}$, where $c^i_{max} = \mathbf{max}\{c_l | l \in \mathcal{L}_i\}$. The strategy space for the game is defined as the Cartesian product $S = \bigotimes_{i=1}^{N} S^{(i)}$, which is equivalent to the feasible set Λ. Strategy $\mathbf{x} = (x_1, x_2, ..., x_N) \in \Lambda$ is called a strategy profile. Each player (e.g. player i) chooses the sending rate (x_i) in the feasible set in order to maximize its *own* payoff function $B_i(x_i)$ in a selfish way. By "in a selfish way" we mean that the player does not care about other players' payoff, as far as the rate vector is in the feasible set.

One of the key questions in a non-cooperative flow control game in general, and our game in particular, is whether the network converges to (or settles at) an equilibrium point, such that no player can increase its payoff by adjusting its strategy unilaterally. In the game-theory terminology such a point is called a *Nash equilibrium*. The Nash equilibrium in our game also reflects the *balance* of the gain and the cost for each player as well as for the network as a whole. A non-cooperative game may have no Nash equilibrium (in its pure strategy space), multiple equilibria, or a unique equilibrium. As for the TCP Vegas game, we can prove the following theorem:

Theorem 4.2.1. *There exists a* unique *Nash equilibrium (in its pure strategy space) for the TCP Vegas game described above.*

Proof
First, let us consider the existence of the Nash equilibrium for the TCP Vegas game. Notice that the feasible set $\Lambda = \{\mathbf{x} | \mathbf{R}\mathbf{x} \leq \mathbf{c}, \mathbf{x} \geq \mathbf{0}\}$ is a nonempty, convex and compact set. It is nonempty because $\mathbf{x} = (\epsilon, \epsilon, ..., \epsilon) \in \Lambda$, where $0 < \epsilon < \frac{c_{min}}{N}$, $c_{min} = \mathbf{min}\{c_l | l \in \mathcal{L}\}$. It is bounded because $x_i \leq c_{max}, i \in \mathcal{N}$, where $c_{max} = \mathbf{max}\{c_l | l \in \mathcal{L}\}$. Assume that $\mathbf{x}_1, \mathbf{x}_2 \in \Lambda$ and $0 < \rho < 1$, we have:

$$\rho\mathbf{x}_1 + (1-\rho)\mathbf{x}_2 \leq \mathbf{R}(\rho\mathbf{x}_1 + (1-\rho)\mathbf{x}_2) \leq \mathbf{c}$$

This result implies the convexity of Λ.

Now let us consider the payoff functions of the players. Notice that $B_i(x_i)$ is a concave function of x_i. Indeed:

$$B_i^{"}(x_i) = -\frac{\alpha_i}{x_i^2} - \sum_{l \in \mathcal{L}_i} \pi_l^{'} < 0 \tag{4.7}$$

From what have been discussed so far, our game has the following properties:

1. The joint strategy space is nonempty, convex and compact.
2. The payoff function of each player is concave in its own strategy space.

According to Theorem 1 in [Ros65], there exists a Nash equilibrium in its pure strategy space.

For the uniqueness of the Nash equilibrium, let's consider the (nonnegative) weighted sum of the payoff functions:

$$\sigma(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{N} w_i B_i(\mathbf{x}), \qquad w_i \geq 0 \tag{4.8}$$

Denote $g(\mathbf{x}, \mathbf{w})$ the pseudo-gradient of $\sigma(\mathbf{x}, \mathbf{w})$, then the Jacobian of $g(\mathbf{x}, \mathbf{w})$ with respect to $\mathbf{x}$ can be computed as follows:

$$\mathbf{G} = \begin{pmatrix} B_{11} & B_{12} & \dots & B_{1N} \\ B_{21} & B_{22} & \dots & B_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ B_{N1} & B_{N2} & \dots & B_{NN} \end{pmatrix}$$

where

$$B_{ij} = \begin{cases} w_i(-\frac{\alpha_i}{x_i^2} - \sum_{l \in \mathcal{L}_i} \pi_l^{'}) < 0 & j = i \\ -w_i \sum_{l \in \mathcal{L}_{(i,j)}} \frac{\partial \pi_l}{\partial x_j} < 0 & j \neq i, \mathcal{L}_{(i,j)} \neq \emptyset \\ 0 & j \neq i, \mathcal{L}_{(i,j)} \equiv \emptyset. \end{cases}$$

where $\mathcal{L}_{(i,j)} = \mathcal{L}_i \bigcap \mathcal{L}_j$. The matrix $\mathbf{G}$ defined above is thus negative definite. As a result, according to Theorem 6 in [Ros65], $\sigma(\mathbf{x}, \mathbf{w})$ is diagonally strictly concave. According to Theorem 2 in [Ros65], the equilibrium point of the TCP Vegas game is unique. □

Remark 4.2.2. *To reach this equilibrium, [Ros65] shows that each player can change its own strategy at a rate proportional to the gradient of its payoff function with respect to its strategy and subject to constraints. This method is in fact equivalent to the gradient projection algorithms described in [LoLa99].*

Remark 4.2.3. *The authors of [LoLa99], using optimization framework, also showed that, under certain assumptions on the step size, these algorithms converge to a system wide optimal point (which is also proved to be unique). Furthermore, it is proved in [LPW02], [Low03] that the rate control of TCP Vegas/Drop Tail and TCP Vegas/REM is indeed based on these algorithms. This implies that* the TCP Vegas game described above converges to a unique Nash equilibrium that is system wide optimal.

4.2.2 Game 2: Parameter Setting of TCP Vegas

In this game, we consider the parameter setting of TCP Vegas. As described in [BMP94], TCP Vegas tries to maintain the number of backlogged packets in the network between α and β. We examine here the situation when a selfish (and greedy) user tries to increase the number of its backlogged packets in the network in order to grab more bandwidth in the

network. If all other players do the same thing (i.e. they are also selfish and greedy), the total number of packets in the network would increase without bound. However, the size of the buffers at routers are bounded and packet loss would occur, reducing the goodput of the connection. We are interested in a situation (i.e. a parameter setting, if at all exists) from where no player would deviate.

We consider a simple topology of N TCP Vegas sources sharing a *single* bottleneck link with a buffer size of B packets. Source i is associated with a set (α_i, β_i). In this dissertation, we deal with the case when $\alpha_i = \beta_i$. The case when $\alpha_i \neq \beta_i$ is left for future work.

Players: N TCP Vegas flows

Actions: Each player can set its parameter (α_i) in order to control the number of its backlogged packets in the queue of the bottleneck link (with capacity μ and delay τ). The router is assumed to use Drop-Tail mechanism (FIFO principle)

Payoff: $f(\alpha_i) = \lambda_i$ (the goodput)

If the total number of backlogged packets is smaller than the buffer size at the bottleneck router (i.e. $\sum_{j=1}^{N} \alpha_j < B$) then the payoff function of player i can be expressed as follows:

$$f(\alpha_i) = \lambda_i = \frac{\alpha_i}{\frac{\sum_{j=1}^{N} \alpha_j}{\mu}} = \frac{\mu\alpha_i}{\sum_{j=1}^{N} \alpha_j} = \frac{\mu\alpha_i}{\alpha_i + \sum_{j\neq i} \alpha_j} \tag{4.9}$$

From Equation 4.9 we have:

$$\frac{\partial f}{\partial \alpha_i} = \frac{\mu \sum_{j\neq i} \alpha_j}{(\alpha_i + \sum_{j\neq i} \alpha_j)^2} > 0, \qquad i = \overline{1 \ldots N} \tag{4.10}$$

Since $\sum_{j\neq i} \alpha_j$ is always positive, it follows from Equation 4.9 that $\frac{\partial f}{\partial \alpha_i} > 0, \forall i$. This implies that given other players' strategies, player i will set α_i as high as possible in order to maximize its payoff. Notice that Equation 4.9 is valid only if $\sum_{j=1}^{N} \alpha_j < B$. Otherwise, TCP Vegas, according to [Bon98], behaves exactly like TCP Reno. In this case, there are two possibilities [Bon98]:

$$\lambda_i^{Reno} = \begin{cases} \frac{(1-\gamma^2)(\omega+1)^2}{2(1-\gamma)(\omega^2+\omega)+1} \frac{\mu}{N} < \frac{\mu}{N} & \text{if } \omega \geq \frac{\gamma}{1-\gamma} \\ \frac{\mu}{N} & \text{otherwise.} \end{cases} \tag{4.11}$$

Thus, we have two cases:
Case 1: $w < \frac{\gamma}{1-\gamma}$
It is important to note that in this case, the link is fully utilized both for TCP Vegas and TCP Reno. Furthermore, in TCP Reno style performance, the bandwidth is fairly (equally) shared between flows (because they have the same RTT). Denote $\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_N^*)$ be the Nash equilibrium of the game in this case. Without losing generality, we can assume that $\alpha_1^* \leq \alpha_2^* \leq \ldots \leq \alpha_N^*$. Notice that in Nash equilibrium, we must have $\alpha_1^* = \alpha_2^* = ... = \alpha_N^*$.

Otherwise, player 1 has the incentive to deviate (i.e. to increase its number of backlogged packets - α_1) in order to get higher goodput, because in Reno style performance, it would get a fairer share of the total bandwidth (i.e. $\frac{\mu}{N}$). As a result, we have the Nash equilibria for this game: $\alpha^* = (\alpha_1^*, .., \alpha_N^*)$ where $\alpha_i^* \geq \lfloor \frac{B}{N} \rfloor, \quad \forall i$. This means that, in this case, in Nash equilibrium, the parameter α can be arbitrarily large.

Case 2: $w \geq \frac{\gamma}{1-\gamma}$

In this case, the link is not fully utilized. Following similar reasoning as in Case 1, we have a set of Nash equilibria defined as follows: $\Omega = \{\alpha = (\alpha_1, ..\alpha_N) | \alpha_1 \leq \alpha_2 \leq .. \leq \alpha_N\}$ with the conditions that $\sum_{i=1}^{N} \alpha_i = B - 1$ and $\alpha_1 \geq \frac{(1-\gamma^2)(\omega+1)^2}{2(1-\gamma)(\omega^2+\omega)+1} \frac{B-1}{N}$. The latter expression simply means that even player 1 (who gets the smallest bandwidth) would not deviate, so no other player would deviate. If this condition does not hold, player 1 would deviate to get higher bandwidth share.

Our final comment on these Nash equilibria is that each TCP Vegas flow (player) maintains the number of its own backlogged packets as many as possible. As a result, the buffer is nearly full and the queueing delay is unnecessarily high. A nearly full buffer may cause many difficulties for TCP Vegas (e.g. the estimation of $baseRTT$ might be inaccurate if there are already many packets in the queue when the connection starts).

4.2.3 Game 3: Application to FAST TCP

In this section, we discuss how to apply the TCP Vegas games investigated above to analyze FAST TCP game. First, let us consider the FAST TCP's window dynamics as described in [JWL04]:

$$w \leftarrow min\{2w, (1-\gamma)w + \gamma(\frac{baseRTT}{RTT} w + \alpha(w, qdelay))\} \tag{4.12}$$

with $0 < \gamma \leq 1$ and the parameter $\alpha(w, qdelay)$ is defined as a function of w and $qdelay$ as follows:

$$\alpha(w, qdelay) = \begin{cases} aw & \text{if } qdelay = 0, \\ \alpha & \text{otherwise.} \end{cases} \tag{4.13}$$

We can interpret FAST TCP as a "faster" TCP Vegas as follows. First, denote $\mathsf{diff} = \frac{RTT - baseRTT}{RTT} w$, we have:

$$w(t+1) = \begin{cases} w(t) + \gamma(\alpha - \mathsf{diff}) & \text{if } \mathsf{diff} < \alpha, \\ w(t) - \gamma(\mathsf{diff} - \alpha) & \text{if } \mathsf{diff} > \alpha, \\ w(t) & \text{otherwise.} \end{cases} \tag{4.14}$$

From Equation 4.14 we can see that FAST TCP increases (or decreases) its window size by $\gamma(\alpha - \mathsf{diff})$, instead of 1 as in TCP Vegas. Since in the original paper [JWL04], no guidance or suggestion on how to set the parameter α as well as other parameters (a, γ)

were provided, we will discuss some of the issues here. First, notice that if α, γ are chosen so that $\gamma(\alpha - \mathsf{diff}) = 1$, then FAST TCP would behave *exactly* like TCP Vegas. In fact, this feature (the *amount* of window increment/decrement) is the major difference between FAST TCP and TCP Vegas. If there is a difference between α and diff, FAST TCP tries to balance this difference by increment/decrement its current window size more quickly than TCP Vegas. So purposely, FAST TCP would set its parameter α much larger than 1 in a very high bandwidth-delay network. This is only from the point of a *single* connection in the network. We will show that, in the case of multiple FAST TCP flows sharing a very high bandwidth-delay product link, from game theoretic point of view, each flow has the incentive to increase its own α_i parameter in order to have better share of bandwidth. Let's consider the following game:

Game 3.1

Players: N FAST TCP flows

Actions: Each player can set its parameter (α_i) in order to control the number of its backlogged packets in the queue of the bottleneck link (with capacity μ and delay τ). The router (with buffering capacity of B packets) is assumed to use Drop-Tail mechanism (FIFO principle)

Payoff: $f(\alpha_i) = \lambda_i$ (the goodput)

Notice that the goodput of a FAST TCP connection has the same formula as of a TCP Vegas connection. That is, if $\sum_{j=1}^{N} < B$, then:

$$\lambda_i = \frac{\alpha_i}{q_i} = \mu \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j} \tag{4.15}$$

Again, similar to the TCP Vegas game, we have the payoff function of player i is strictly increasing function with respect to α_i. This implies that the FAST TCP game is similar to the TCP Vegas game when the bandwidth-delay product is large (Game 2.1, Case 2) As a result, the Nash equilibrium for the FAST TCP game is the same as in TCP Vegas game (the only difference is the *rate* to equilibrium, not the equilibrium itself). There is a number of problems associated with these Nash equilibria. First, the nearly full queue-length at equilibrium makes the network vulnerable in the sense that there is a high probability of FAST TCP flows' behaviour turns back to the behaviour of TCP Reno and we come back to where we started. Second, a large queue-length means a high delay, which is undesirable.

4.3 Conclusion

We have demonstrated, by using game-theoretic approach, how TCP Vegas' inherent pricing schemes as well as the parameter setting impact on its performance. Our analysis shows

that these inherent pricing schemes result in a rate control equilibrium state that is a Nash equilibrium in game-theoretic terms which is also a global optimum of the all-Vegas networks. We also proved that the parameter setting of TCP Vegas (and also FAST TCP) are very vulnerable to selfish actions of the users. This poses a serious threat to the possible deployment of FAST TCP in the future Internet.

Chapter 5

Performance Analysis of RED

In the first part of the dissertation (Chapter 2, Chapter 3 and Chapter 4), a comprehensive performance analysis of TCP is presented. The TCP protocol represents the end-point control of the traffic. However, the traffic over the Internet also depends on other mechanisms of the network (such as buffering and routing). As a result, the traffic control mechanisms of TCP are not enough to control the traffic over the Internet and they must be supplemented by mechanisms inside the networks. Among those mechanisms are the queue managements at the routers. The classical Drop Tail scheme does little in this respect. Recently, a class of Active Queue Management (AQM) schemes has been proposed to enhance the performance of the network. The Random Early Detection (RED) mechanism is of particular interest because it is already implemented in a wide range of commercial routers (such as Cisco's routers). However, the performance of RED as well as the tuning of RED's parameters are still very problematic and open issues.

There exists a substantial literature on performance analysis of RED and it can be categorized into two main classes. The first class largely deals with analyzing and configuring RED while keeping the algorithm intact. The second class considers how to change the original RED to have better performance. In fact, there is no distinct border between the two classes. In respect to analyzing RED, May *et al* [MBB00] proposed a simple analytic model of RED and concluded (among others) that RED, in certain circumstances, provides no better performance than Drop Tail. Christiansen *et al* in [CJOS00] evaluated RED with pure web traffic and concluded that RED offers no clear advantage over Drop Tail, at least in terms of delay. The paper also reports that performance is quite sensitive to the setting of RED parameters. Problems with tuning and configuring RED parameters can also be found in [MGT00, HMTG01]. In respect to new modification to RED, we would mention Self-Configuring RED in [FKSS99] and recently Adaptive RED in [FGS01]. Basically, the authors propose adapting the dropping probability max_p as a function of average queue size to achieve the specified target average queue size in a wide variety of traffic scenarios. We argue, however, that the adaptation of any parameter will affect the overall system performance. We see no clear justification for adapting only max_p rather than min_{th} and max_{th}, as long as $min_{th} < max_{th} < K$. In Sally Floyd's Adaptive RED, the authors proposed the tuning of w_q based on link capacity. However, what we really consider here

is *available* capacity, which is changing, and the dynamics of which is yet to be estimated. Other modifications to RED can be found in [ZhAt00, LiMo97, AOMC01, WyZu02].

Despite the fact that extensive research has been devoted to performance analysis of RED and many publications have highlighted various aspects of this issue, the question of how to configure the parameters of RED for optimal performance is still open. In addition, the impact of RED mechanism on different issues of Internet performance (such as fairness) is still not clear and requires more clarification and analysis.

In this chapter, a comprehensive performance analysis of RED is presented. We revisit some features in RED and study them in greater detail. We point out that RED, in general, does not possess proportional loss between flows as claimed and widely adopted in previous research. We suggest the generalization of the Poisson Arrivals See Time Average (PASTA) property and give a proof for TCP flows. We also evaluate the performance of the Exponential Weighted Moving Average (EWMA) algorithm in RED. We find that EWMA in RED is an unbiased estimator of the average queue-length, regardless of the weighting value w_q. We also point out the theoretical and practical limits of EWMA in RED. Finally, we propose the use of Fuzzy EWMA to RED (Fuzzy RED) to alleviate the inflexibility of RED tuning. We use simulations to evaluate the performance of Fuzzy RED and to compare it with other versions of RED. Our simulations show that, in the case of a high workload and a high level of variation, Fuzzy RED, by tracking system variation in an on-line manner, improves RED performance in a number of important router-based metrics like packet loss rate, average queueing delay, link utilization, and global power.

The chapter is organized as follows. In Section 5.1, the background on RED is briefly overviewed. In Section 5.2, we give a detailed analysis of proportional loss in RED. Section 5.3 shows the simulation topology. Section 5.4 discusses the motivation for Fuzzy RED. Section 5.5 describes the Fuzzy RED mechanism. Section 5.6 evaluates the performance of Fuzzy RED. Finally, Section 5.7 concludes the chapter.

5.1 Background on RED

In this section, we briefly describe the elements of the RED mechanism. The details about the RED scheme can be found in the original paper [FloJa93]. The first key difference between RED scheme and the Drop Tail scheme is that the RED scheme starts dropping packets *early*, before the queue is full. The second key difference is that the dropping decision of the RED scheme does not depend on the *instantaneous* queue-length but depends on the *average* queue-length instead. Basically, the RED scheme *randomly* discards an incoming packet with a probability that is proportional to the average queue-length.

Specifically, the RED scheme uses an exponential weighted moving average (EWMA), which is a sort of a low-pass filter, to calculate the average queue-length from the current queue-length:

$$a\hat{v}g_t = w_q q_t + (1 - w_q)a\hat{v}g_{t-1} \tag{5.1}$$

where q_t is the instantaneous queue length at time t and $w_q \in [0, 1]$ is the weighting value.

Using the average queue length, the RED scheme calculates a packet marking/dropping probability p_b at every arrival of an incoming packet as follows:

$$p_b = \begin{cases} 0 & \text{if } a\hat{v}g < min_{th}, \\ 1 & \text{if } a\hat{v}g \geq max_{th}, \\ \frac{a\hat{v}g - min_{th}}{max_{th} - min_{th}} * max_p & \text{otherwise.} \end{cases} \tag{5.2}$$

where max_p denotes the maximum packet marking probability, min_{th} denotes the minimum threshold, max_{th} denotes the maximum threshold and B denotes the buffer size. Figure 5.1

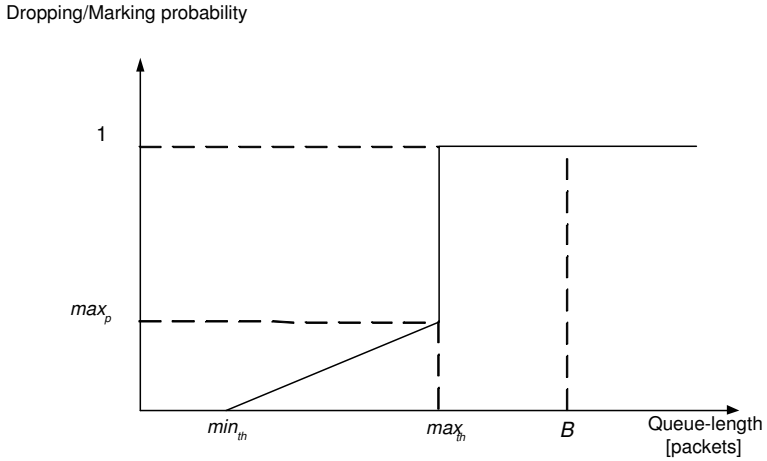


Figure 5.1: Dropping function of RED

illustrates the dropping function of the RED mechanism. As we can see in Figure 5.1, when the (estimated) average queue-length is less than the min_{th}, no packet is marked (dropped). However, if the average queue-length is between the min_{th} and max_{th} then the marking probability is a linear function of the average queue-length. If the average queue-length is greater than max_{th}, then all the incoming packets are marked (dropped). The operation algorithm of the RED scheme discussed so far is relatively simple. As a result, it is relatively easy to implement in commercial routers. In fact, the RED schemes has been widely deployed in a number of commercial routers, including the Cisco's routers.

5.2 Proportional Loss Revisited

Loosely speaking, the proportional loss property means that the fraction of marked packets for each connection is proportional to that connection's share of the bandwidth. RED is claimed to possess this property [FloJa93]. In addition, proportional loss is widely adopted (e.g. [LiMo97, HBT99]) in the fairness analysis of RED. However, M. May *et al* in [MBB00]

suggested that the claim is true *only if* the arrival flows are Poisson arrivals. This is based on the PASTA property of Poisson processes. We take one step further. Notice that PASTA can be generalized to ASTA (Arrivals See Time Averages), [MeYa95], and Burke in [Bur76] has shown that the composite stream of exogenous Poisson arrivals and feedback customers is *not* Poisson even though this stream sees a time average. Since TCP flows account for a large portion of Internet traffic, TCP arrivals are mainly of interest. The question that arises is then: Do TCP arrivals see time averages or not?

Proposition 5.2.1. *TCP arrivals do not see time averages either with RED, or with Drop Tail.*

Proof
Let $N \equiv \{N(t), t \geq 0\}$ be the queue length process and $A \equiv \{A(t), t \geq 0\}$ be the arrival process. For an arbitrary set B in the value space of N, define

$$U(t) = \begin{cases} 1 & \text{if } N(t) \in B \\ 0 & \text{otherwise} \end{cases}$$

If B is the stationary queue-length, then U is the event that N remains in that state. Now, let us consider the mechanism of TCP. For the sake of simplicity, we take TCP Reno for our analysis. Let W be the congestion window size and W_{th} the threshold value. Notice that if the sender always has data to send then the congestion window is approximately the number of packets that were sent but not yet acknowledged. In TCP/IP networks, the acknowledgements (ACKs, or *feedbacks*) may traverse through the same route as the data packets, they may traverse in different route. In respect to the first case, the congestion window directly reflects the dynamics of the packet flow feeding the router. In respect to the second case, the number of packets going in a forward direction, in a stable period, is approximately half of this value (since the other half are ACKs in the backward direction). Consequently, the dynamics of the congestion window also reflect the dynamics of the packet flow feeding the router of interest.

1. After every nonrepeated acknowledgment: if $W < W_{th}$, set $W = W + 1$; *Slow Start Phase* else set $W = W + 1/W$; *Congestion Avoidance Phase*

2. **When the duplicate acknowledgments exceed a threshold, retransmit next expected packet; set** $W_{th} = W/2$**, then set** $W = W_{th}$ **and enter** *Fast Recovery Phase*

3. Upon timer expiration, the algorithm goes into slow start: set $W_{th} = W/2$ set $W = 1$.

Let us consider Phase 2, when the congestion window is halved after sensing duplicate acknowledgements. Duplicate ACKs imply dropping of packets at the buffer and that the buffer at the router is full (for Drop Tail) or potentially full (for RED). That is, the future

increments of A in this phase are *dependent* on the past of U. And so, the ASTA property does not hold. Consequently, TCP arrivals do not see time average either with RED, or with Drop Tail gateway. □

Corollary 1. *We cannot achieve proportional loss between TCP flows either with RED or with Drop-Tail.*

Remark 5.2.2. *A more general condition of ASTA is LBA [MeYa95](Lack of Bias Assumption) which only requires that U and the conditional intensity, η_U, of N, given U, are point-wise uncorrelated. Certainly the uncorrelated condition is weaker than the independence condition. However, we can similarly show that this condition also fails.*

Remark 5.2.3. *ASTA, in the absence of Poisson flows, are all in the category of networks of quasi-reversible queues; in particular, for the M/M/1 queue with feedback. Once again, Burke in [Bur76] has shown that the composite stream of exogenous Poisson arrivals and feedback customers is not Poisson even though this stream sees a time average.*

Remark 5.2.4. *It is noteworthy, however, that for quasi-reversible queues* **in isolation**, *LBA implies Poisson arrivals [MeYa95].*

Remark 5.2.5. *Let us assume that the service time at the router is exponentially distributed (Markovian service). In this case, consider the G/M/1 queue. We allow the arrival process to be general. Certainly, the arrivals generally (except Poisson ones) do not see time averages, but due to the duality of M/G/1 and G/M/1, we can explicitly express these two values by each other [Kle75].*

5.3 Simulation Topology

Figure 5.2 shows the topology template for all of our simulations throughout this chapter. We consider the general topology of N senders $S_1, S_2, .., S_N$ and N access links. The i-th access link is specified by bandwidth B_{Si} and delay D_{Si}. Router1 is the access router. We suppose the link between Router1 and Router2 is a bottleneck link with bandwidth B_{BN}

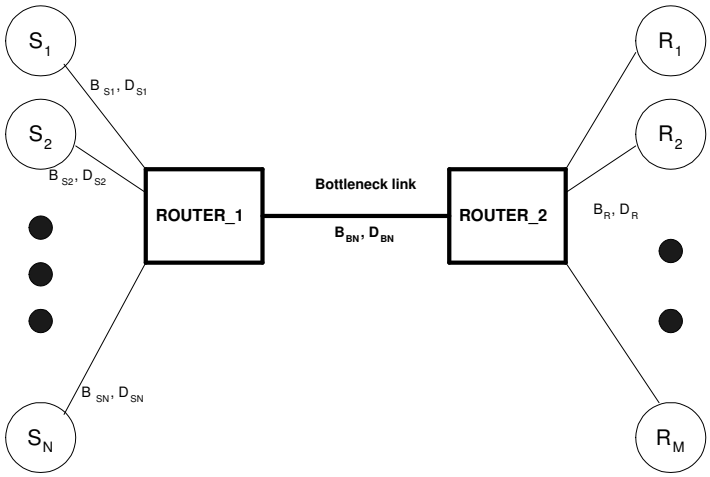


Figure 5.2: Simulation Topology

and delay D_{BN}. We also add M receivers at the other end in case we want to generate backward traffic. However, unless otherwise stated, we consider the bottleneck link is the only sink.

5.4 Motivations for Fuzzy RED

5.4.1 Pitfalls in Tuning RED Parameters

One of the inherent weaknesses of RED is parameter sensitivity. Extensive research has been devoted to this issue and many publications have highlighted various aspects of this issue. However, the question of how to configure the parameters of RED for optimal performance is still open. Christiansen *et al* in [CJOS00] examined the impacts of tuning RED's parameters on end-user *delay*, and concluded that for links carrying only web traffic, RED queue management appears to provide no clear advantage over the Drop-Tail gateway for end-user response times. M. May *et al* in [MBB00] used a simple analytic model to evaluate RED performance in terms of *loss rate, link utilization, delay* and *delay variation.*

In this section, we use simulations to examine the impacts of tuning different RED parameters and compare their performance with Drop-Tail. We concentrate on three router-based metrics: *link utilization*, *link loss rate* and *average queuing delay.* We believe that these metrics clearly provide insight into the performance of queueing management algorithms at routers because end-user metrics of interest (such as end-user delay) are mainly dependent on these metrics. Our simulations reveal two main points. First, RED with fixed, default parameters is no better than Drop-Tail, at least in terms of the examined metrics. Second, there exist parameter tunings of RED so that they can perform somewhat better than Drop-Tail. However, these parameter settings do not increase RED performance both

in link utilization and average queuing delay simultaneously. Rather, in this case, RED performs better than Drop-Tail in terms of *global power* defined in [FloJa93] as the ratio of throughput to delay.

Impact of weighting parameter w_q. We examine the impact of tuning the weighting parameter w_q when other parameters are left unchanged and equal to the default values (max_p=0.1, min_{th}=10 packets, max_{th}=30 packets according to the buffer size of 50 packets). The bottleneck link bandwidth is 15 Mb/s, with delay 50 ms. The access links are all 100 Mb/s. All connections are TCP connections with a packet size of 1000 bytes. To simulate the impact of w_q on different workloads, we examine it with an increasing number of connections (4, 16, 64, 256, accordingly). Increasing the number of connections means increasing the workload feeding the router at the bottleneck link. To simulate high level of variation of incoming TCP traffic, we set the access link delays in a range from 10ms to $(10 + N - 1)$ ms, where N is the number of connections (nodes). The simulation time used is 30 seconds. We experience a large fluctuation in queue length dynamics in the first few seconds (typically around 5 seconds in our case) due to TCP's first slow starts. So, the simulation time should not be too short. We find that 30 seconds simulation time is adequate to ensure statistical accuracy and to match TCP session time details.

RED is tuned according to the recommendation found in [FGS01]. We set

$$w_q = 1 - exp(-1/B_{BN}) \tag{5.3}$$

where B_{BN} is the bottleneck link capacity. In our simulation, B_{BN} is set to 15 Mb/s, so w_q is set to 0.005, accordingly.

Figure 5.3(a) shows the impact of tuning w_q on link loss rates. As we can see, Drop-Tail performs better than the default tuning of RED, at least in terms of link loss rates. We observe that as the number of connections is small and the round-trip times are relatively in the same range, link loss rates are relatively similar. However, as the number of connections increases, the difference becomes significant. The simulation results reveal to us that there exists a parameter tuning for RED that produces better performance than Drop-Tail. However, Drop-Tail seems to be more robust than a number of cases with RED, especially when max_{th} is far from buffer size and max_p is high (aggressive early detection). Figure 5.3(c) shows the performance of RED and Drop-Tail in terms of link utilization. We observe that default RED is rather aggressive in detection, thus reducing the utilization of link capacity, especially when the workload is high (increased connection number).

As expected, the results are different with delay. Figure 5.3(b) shows that both versions of RED (default and tuned) have smaller average queueing delay than Drop-Tail. However, we have to find the trade-off between average queueing delay and link utilization. We use global power to judge the trade-off performance of Drop-Tail and different parameter settings of RED. Figure 5.3(d) shows that RED indeed performs better than Drop-Tail in terms of global power. However, this metrics is hardly observable by the end-user.

Impact of dropping parameter max_p. We examine the impact of tuning the dropping parameter max_p when other parameters are left unchanged and equal to the default value(w_q=0.02, min_{th}=10 packets, max_{th}=30 packets according to the buffer size of 50 packets). The simulation topology is the same as the simulation with w_q. The bottleneck

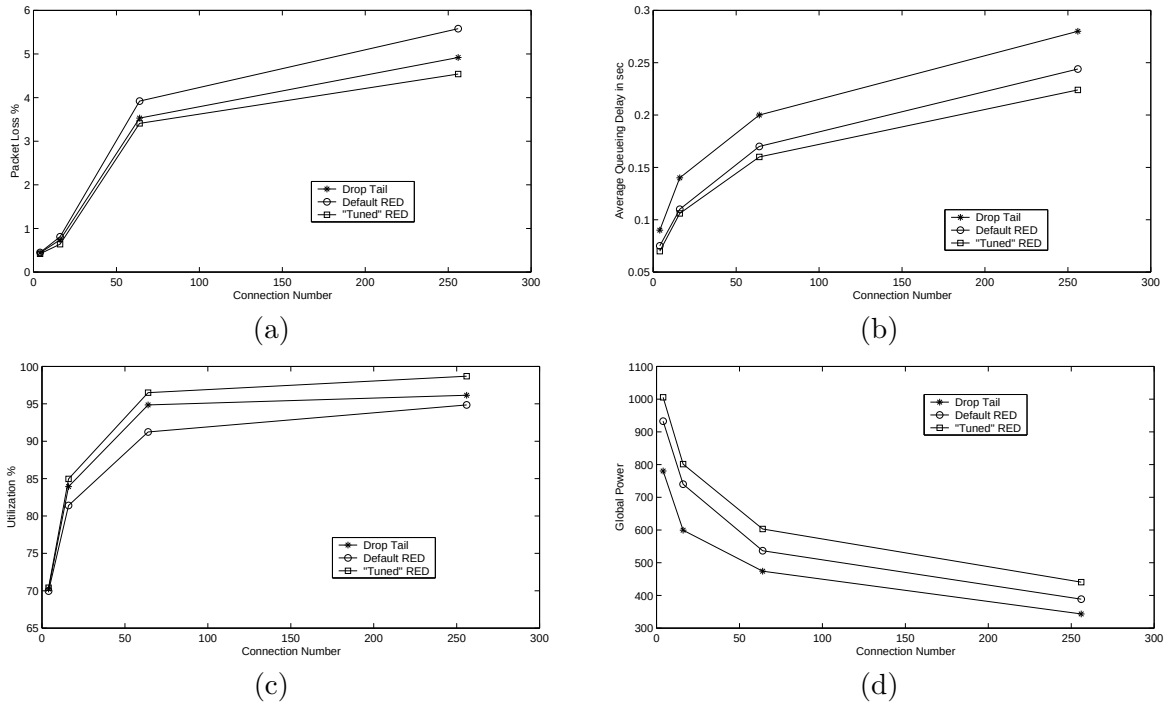


Figure 5.3: Impact of weighting parameter on router-based performance metrics: RED vs. Drop-Tail

link bandwidth is 15 Mb/s, with a delay of 50 ms. The access links are all 100 Mb/s. All connections are TCP connections with a packet size of 1000 bytes. To simulate the impact of max_p on different workloads, we examine it with an increasing number of connections (4, 16, 64, 256, accordingly). To simulate a high level of variation of incoming TCP traffic, we set access link delays ranging from 10ms to $(10 + N - 1)$ ms, where N is the number of connections (nodes). The simulation time is 30 seconds.

RED is tuned according to the recommendation in [FKSS99], with α and β are set to 3 and 2, respectively.

As we can see in Figure 5.4(a) and 5.4(b), we have similar results in terms of loss rates and delay as with the w_q tuning simulations. However, Figure 5.4(c) shows that tuning max_p according to Adaptive RED in [FKSS99], unlike the w_q tuning simulations, also improve link utilization. Power, as a ratio of throughput to delay, is thus certainly improved.

What we can conclude here is that, fixed, default RED shows no clear advantage over Drop-Tail in a number of crucial router-based performance metrics. However, there exists a parameter tuning that can improve RED performance. The problem remains here is that, as the conditions are changing, how to adapt the tuning properly to maintain robust performance.

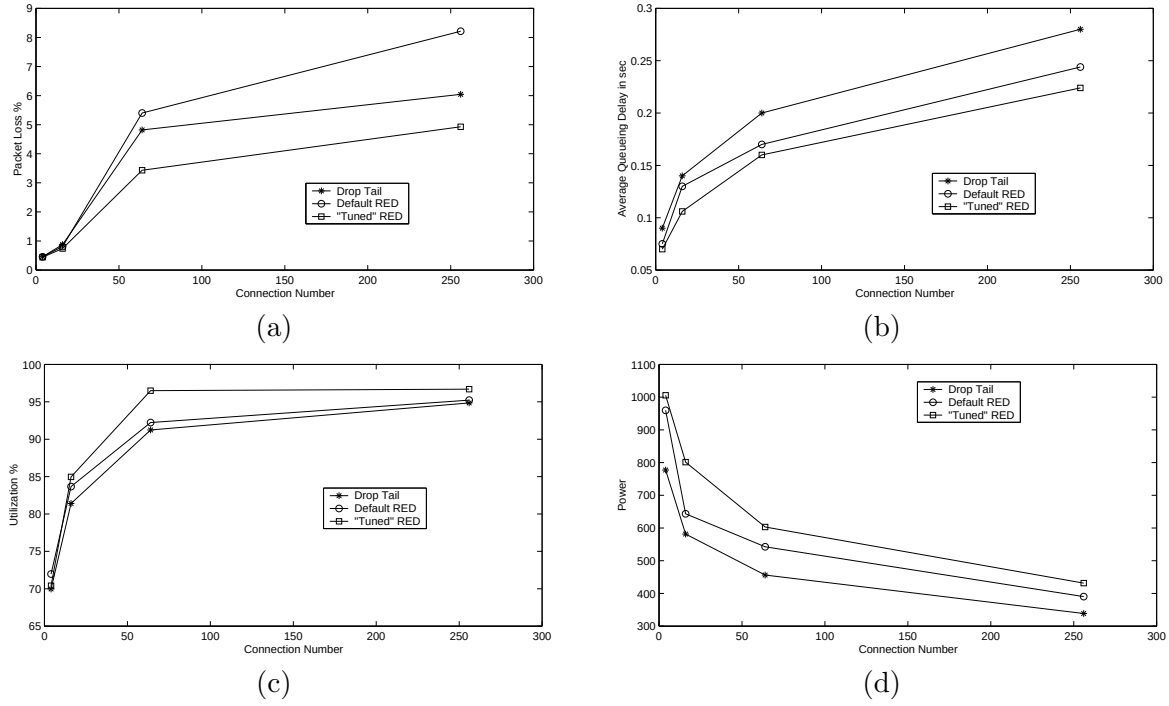


Figure 5.4: Impact of dropping parameter on router-based performance metrics: RED vs. Drop-Tail

5.4.2 Reasons for Fuzzy Extension

Theoretical Limits of EWMA in RED

For any fixed $w_q \in [0, 1]$, let $a\hat{v}g_t$ be the estimator of the average queue length by:

$$a\hat{v}g_t = w_q q_t + (1 - w_q) a\hat{v}g_{t-1} \tag{5.4}$$

where q_t is the instantaneous queue length at time t. Given the estimator, a natural question arises then: Is the estimator an unbiased estimator? The following well-known fact in statistics gives us the answer:

Lemma 5.4.1. *If $\{q_t\}$ is stationary with $E(q_t) = \mu_q$ then $a\hat{v}g_t$ is an* unbiased *estimator of μ_q, regardless of the weighting value w_q.*

Now, let us consider the variance of this estimator. Without losing generality, we can suppose that $q_1 = 0$, that is the queue starts from empty. Let σ^2 be the variance of $\{q_t\}$.

Lemma 5.4.2. *[YMKT99] If $q_1, q_2, ...$ are independent (uncorrelated) then the variance of*

the estimator can be calculated as:

$$D^2(a\hat{v}g_t) = \sigma^2 \frac{w_q - w_q(1-w_q)^{2t-2}}{2-w_q} \tag{5.5}$$

From Equation 5.5, if w_q is small ($w_q \approx 0$) then $D^2(a\hat{v}g_t) \approx \sigma^2 \frac{w_q}{2}$ as $t \to \infty$.

Now consider the case when $q_1, q_2, ..$ are correlated. Denote $\gamma(k) = E\Big[(q_{t+k} - \mu_q)(q_t - \mu_q)\Big]$ the covariance function of $\{q_t\}$ at lag k and $\varrho(k) = \gamma(k)/\gamma(0)$ the correlation function of $\{q_t\}$ at lag k.

Proposition 5.4.3. *The variance of the estimator can be calculated as:*

$$D^2(a\hat{v}g_t) = \sigma^2 \frac{w_q - w_q(1-w_q)^{2t-2}}{2-w_q} + 2\sum_{k=1}^{t-2} \varrho(k) \sum_{j=0}^{t-2-k} w_q^2(1-w_q)^{2j+k} \tag{5.6}$$

This proposition is actually the corollary of Lemma 5.4.1 and Lemma 5.4.2.

Remark 5.4.4. *The coefficient $\varrho(k) \to \frac{w_q}{2-w_q}(1-w_q)^k$ as $t \to \infty$. Interestingly, the correlation function $\varrho(k)$ in the expression is also "exponentially weighted" with the weighting parameter $1 - w_q$.*

Remark 5.4.5. *The additional term contributes to the variance of the estimator. This makes the estimator worse (it is not so good already, compared with a moving window), since it increases the variance of the estimator. In practice, empirical and simulation analysis in [VaFe01] show that the queue-length process is not only correlated, but exhibits fractal properties, e.g. long range dependence. In means that $\varrho(k)$ decays according to a power law. This slow decay in the correlation can make this additional term large in practice.*

Practical Limits of EWMA in RED

The standard Exponential Weighted Moving Average applied in RED possesses a number of good properties. It is easy to be implemented and it requires only a small buffer size for the storage of samples. It is, as mentioned in the previous section, also an unbiased estimator of the mean. However, it is inflexible in some points. First, when we average the queue-length, we are implicitly choosing a *time scale* over which to average it. The problem is then "What should that time scale be?". Intuitively, it should match the round-trip time of a typical TCP connection through the RED buffer. In practice, however, TCP connections

can have round-trip times which vary by several orders of magnitude. Furthermore, TCP is self-clocking and so already has its own averaging mechanism built-in which automatically averages over a round-trip time. So why should we try to average something that is already doing its own averaging and when it's simply impossible to get the time scale right anyway? Second, RED was basically designed to face with *transient congestion* [FloJa93] and *highly periodic* network traffic, especially TCP traffic. In this respect, the standard EWMA gives a *fixed* weight to past history, thus ignoring transient phases in system dynamics. In [FloJa93], the authors proposed an analysis of bounds (or guide-lines) for the weighting value w_q. The analysis in that paper is only for a *given* burst size and buffer size. In other words, we need to know these parameters *a priori* in order to find an appropriate w_q to meet our performance target. A fixed w_q is inflexible in the sense that the EWMA algorithm cannot adapt to the changing condition of the incoming traffic. To alleviate this problem, we propose the use of Fuzzy Exponential Averaging [Kes91], which automatically determines a 'good' value of w_q, and is able to change this value on-line if the system behaviour changes. Since the RED dropping mechanism is based on the estimated average queue-length, with "good value", we mean that RED can better keep track with queue-length variations, and consequently, reduces the number of unnecessarily dropped packets at the router.

5.5 Fuzzy RED Mechanism

We basically keep the RED mechanism intact and only modify the weighting parameter w_q. When estimating the average queue-length at the router, instead of using a fixed weighting parameter, we apply Fuzzy EWMA. Details about Fuzzy EWMA are described in the original paper [Kes91]. Now, we shall discuss how Fuzzy EWMA works in our case.

5.5.1 Construction of Fuzzy RED Mechanism

Consider a discrete time system with q_k, the queue length at the buffer at time k, as the state variable. The system can span a spectrum varying from 'steady' (stationary) to 'noisy' (non-stationary). Let $\hat{q}_k$ be the estimate of q_k, then observation noise (error) is $q_k - \hat{q}_k$. To see the relation between error and the predictor, we define scaled error as $|q_k - \hat{q}_k|/\hat{q}_k$. From now on, if not further mentioned, we deal with this error, because it gives us insight into how the error is related to the estimated queue-length. The variance of system and observation noise is the problem. We need to construct a predictor that can adapt to the changing of system dynamics. We consider the Fuzzy EWMA for this purpose.

The first question we need to deal with is how to define the control rules. We assume that when the queue stays in its stationary (stable) state, the *estimation error* is small. That is, if the dynamics of queue-length in the buffer has little perturbation, then the exponential averaging technique will produce a predictor that is usually close to the actual system state (error is small). In this case, w_q should be large. In contrast, when there is a large variation in queue-length, past history cannot predict the future well (the error is high). In this case,

we set w_q low, so that the estimator can track changes in the system. Finally, since we do not have a good grasp of the state dynamics, we only define three gradations in the values of w_q and *error*. In addition, keeping the number of gradations minimal reduces the overhead computing time for the algorithm. Thus, we adopt the following control rules:

- IF *error* is HIGH THEN w_q is LOW
- IF *error* is MEDIUM THEN w_q is MEDIUM
- IF *error* is LOW THEN w_q is HIGH

Secondly, we need to answer the question: HIGH, MEDIUM, LOW are related to what? The answer for this question is equivalent to defining the membership functions for *error* and w_q. For the sake of simplicity, we use the trapezoid form (the conventional and simplest form) for these two variables.

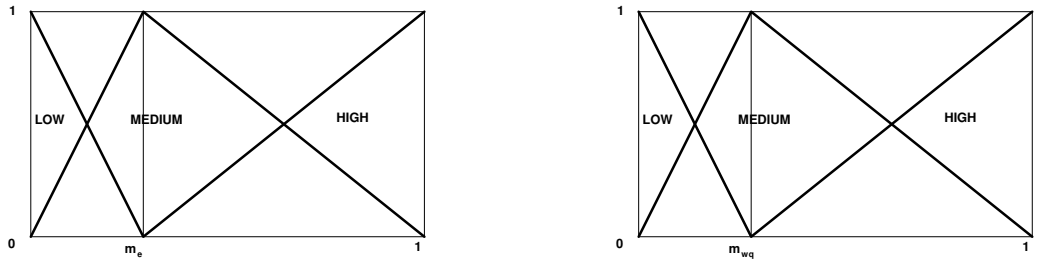


Figure 5.5: Membership functions: (a) error membership function; (b) Wq membership function.

The question that remains is how to specify the membership functions.

How to Specify Membership Functions

This question is equivalent to specifying m_e and m_{w_q}, as shown in Figure 5.5. It can be done on-line by neural network training algorithms (such as back propagation), but this is time consuming and lacks simplicity. So we do the training off-line to find appropriate values for these parameters (i.e. for each topology, we run simulations off-line and let the parameters m_e and m_{w_q} span the whole range. The best values (ranges) are chosen). When it is good, it can be fixed. The outcome of the training, for the topology of our simulations, has m_{w_q} in [0.002..0.05] range and m_e in [0.06..0.2] range. Interestingly, the results for medium value of w_q are close to the value proposed by Sally Floyd *et al* in [FGS01]. At this point, it seems that we arrive at the point where we started. That is, we still need to train the system for some *a priori* knowledge. The only difference here, and also the intuitive force behind our approach, is that once a good trained parameter is chosen, it can be fixed, and from that point, the system will adapt to the changing condition of the incoming traffic.

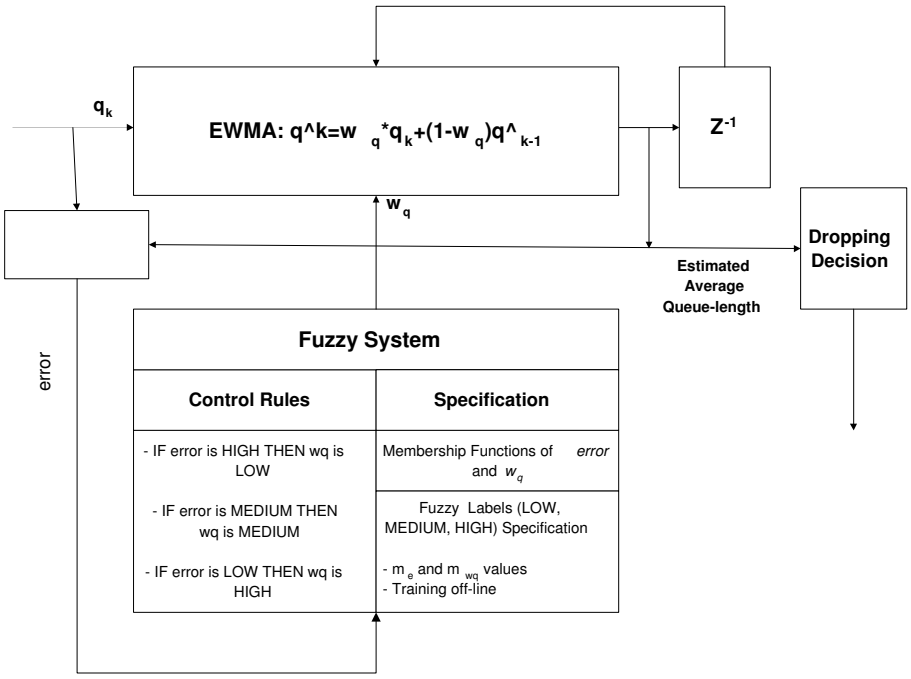


Figure 5.6: Flow diagram of Fuzzy RED

It should be mentioned that we only apply the simplified version of Fuzzy EWMA proposed in [Kes91] without a smoothed proportional error because we find that it is very time consuming and in consequence greatly affects the performance.

The proposed algorithm was implemented in using NS2. Except for the EWMA algorithm part, all other features in RED are kept intact.

5.6 Simulation Results

5.6.1 Stationary Performance

To examine the stationary behaviour of Fuzzy RED, we first run the simulation with the same parameters as in previous Sections. That is, the access links are all 100 Mb/s. The bottleneck link bandwidth is 15 Mb/s, with delay 50 ms. Buffer size at router-1 is set to 50 packets, min_{th} is set to 10 packets, max_{th} is set to 30 packets. Connections are TCP connections with a packet size of 1000 bytes. To simulate the impact of different workloads on performance of versions of RED and Drop-Tail, we examine them with an increasing number of connections (4, 16, 64, 256, accordingly). The simulation time is 30 seconds. We compare our proposed Fuzzy RED not only with Drop Tail and default RED, but also with other Adaptive RED versions, such as Adaptive RED in [FKSS99] (we call it Adaptive

RED-Feng), and Adaptive RED in [FloJa93] (we call it Adaptive RED-Sally).

Scenario 1. *TCP incoming traffic with different RTTs.* To simulate high level of variation of incoming TCP traffic, we set access link delays range from 10ms to $(10+N-1)$ ms, where N is the number of connections (nodes).

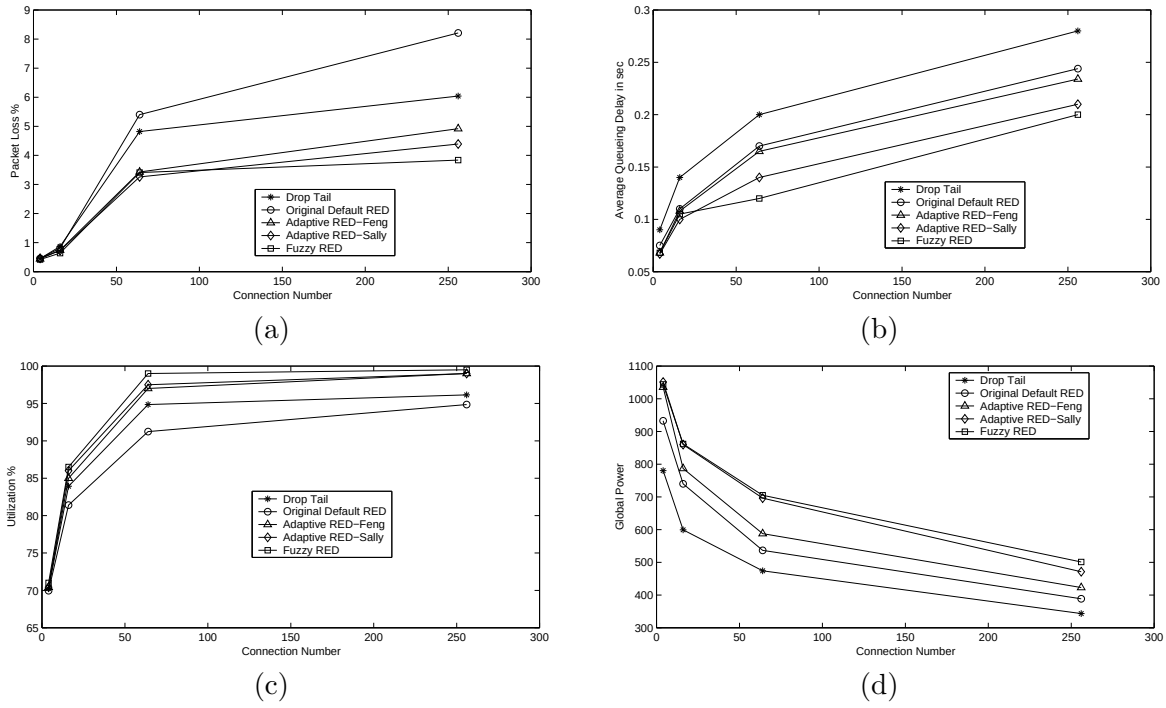


Figure 5.7: Router-based performance metrics- different RTTs: Fuzzy RED vs. Adaptive RED versions and Drop-Tail.

Figure 5.7 shows comparative performance of Fuzzy RED against other versions of RED and Drop Tail. As mentioned and explained in previous sections, we concentrate on router-based performance metrics. We learned from our simulations that under light-weight load (few number of connections), there is no significant difference between versions of RED and Drop Tail, and the orders are changing from simulation to simulation. But the situation is different with heavily loaded incoming traffic (eg. 256 connections). In most of our simulations, three versions of Adaptive RED perform closely together in all examined performance metrics. The benefits of Fuzzy RED are more visible when the workload is high (ie. there are many TCP flows, sufficient training data for the Fuzzy Scheme) and the level of variation of burstiness is high (different round-trip times of TCP connections as in this scenario). Original default RED suffers from a high loss rate because of fixed parameter setting. These fixed default parameters seem to be too aggressive. In terms of loss rate, RED with fixed default parameters, in our case, perform even worse than Drop-Tail. We believe that, this happens because RED, in this case, unnecessarily and too early dropped the incoming packets. Packet loss rates with versions of Adaptive RED in the case of heavy

load (256 TCP flows, with different round-trip time setting) oscillate around 5 percent whereas it is far above for Drop Tail and Default RED (6-10 percent). Figure 5.7(b) shows the comparative performance of the queueing management algorithms in terms of average queueing delay. We experience the situation where Drop Tail performs worst because Drop Tail only drops packets when the queue is full thus keeping the queue potentially full all of the time. One more thing to mention is that RED with fixed default parameters has low utilization as shown in Figure 5.7(c). Interestingly, Figure 5.7(d) reveals that all versions of RED (default RED included) perform better than Drop-Tail in terms of global power as mentioned in previous sections. This means that what we really benefit from RED is not only a low average queueing delay but also the *trade-off* between delay and utilization, at least in terms of global power as defined in [FloJa93].

Scenario 2. *TCP incoming traffic with the same RTTs.* To simulate a low level of variation of burstiness, we set access link delays all equal to 10 ms.

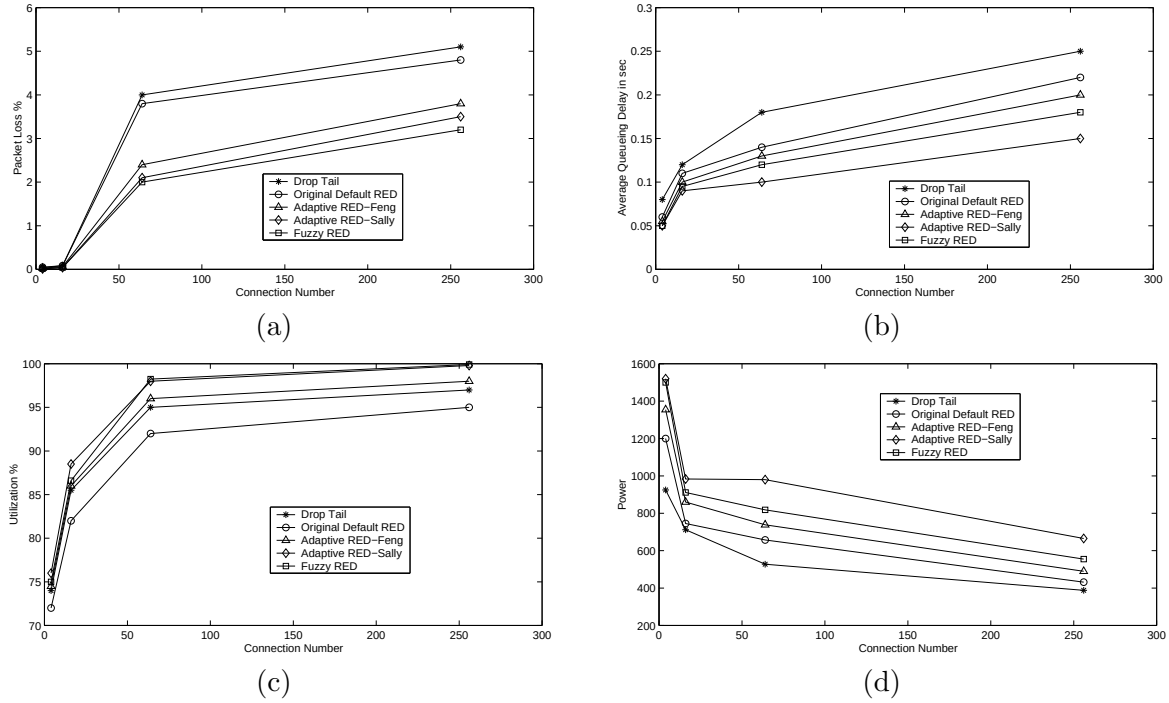


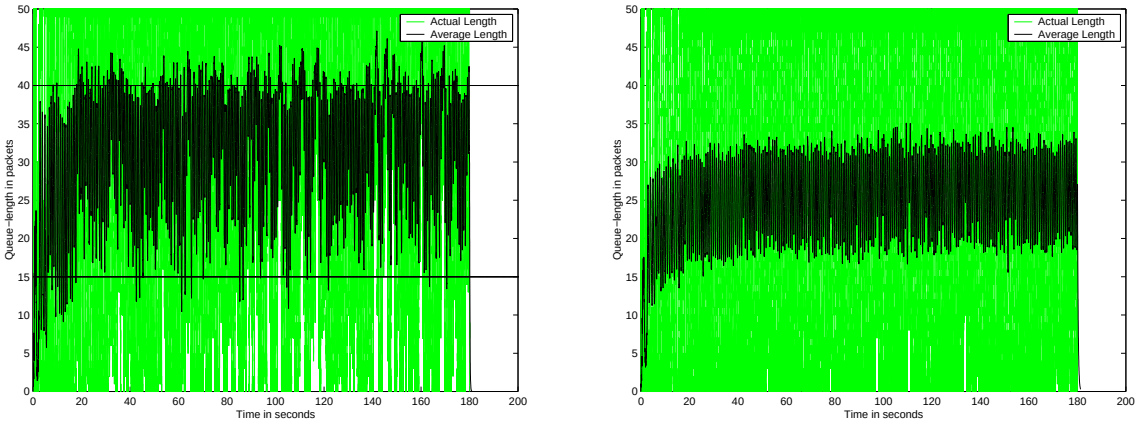
Figure 5.8: Router-based performance metrics- same RTTs: Fuzzy RED vs. Adaptive RED versions and Drop-Tail.

Figure 5.8 shows comparative performance of Fuzzy RED against other versions of RED and Drop Tail. As we can see in Figure 5.8(a), packet loss rates with versions of Adaptive RED in the case of heavy load (256 TCP flows) oscillate around 3 percent whereas it is much higher for Drop Tail and Default RED (4-5 percent), which are significantly smaller than simulation with different RTTs. The situation is similar with delay and link utilization (nearly 100% with Adaptive RED-Sally and Fuzzy RED, 256 connections) in the way that

all the versions perform somewhat better with the same RTTs than with different RTTs, as shown in Figure 5.8(b) and Figure 5.8(c). Global power, as a result and shown in Figure 5.8(d), is certainly improved in all cases. An important observation to be noticed here is that, for this scenario, Adaptive RED-Sally out-performs all other versions of RED and Drop-Tail. This outcome shows the benefit of simplicity in Adaptive RED-Sally compared to Fuzzy RED. However, as we mentioned earlier, we basically design Fuzzy RED to deal with high workload and high level of variations of burstiness which is, we believe, a realistic condition of today's Internet.

What we have been discussing so far is only for pure TCP traffic. However, as RED routers are also responsible for directing and managing other flows of different traffic such as voice and video traffic. For these applications, other performance metrics are also of importance. For example, for VoIP (Voice over IP) applications, not only the average delay but delay variation (jitter) heavily affects the end-user view of performance. So in case both TCP flows and UDP flows sharing the router, queue-length variation should be kept low for the sake of the Quality of Service (QoS) for voice applications. We simulate RED and Fuzzy RED with 1000 flows (500 TCP flows and 500 UDP flows). Since the number of connections in this significantly increased, we also increase the duration time of the simulation to 180 seconds in order to ensure statistical accuracy of our results.

Scenario 3. *Queue-length variation with the same RTTs.* To simulate low level of variation of burstiness, we set access link delays all equal to 100 ms.

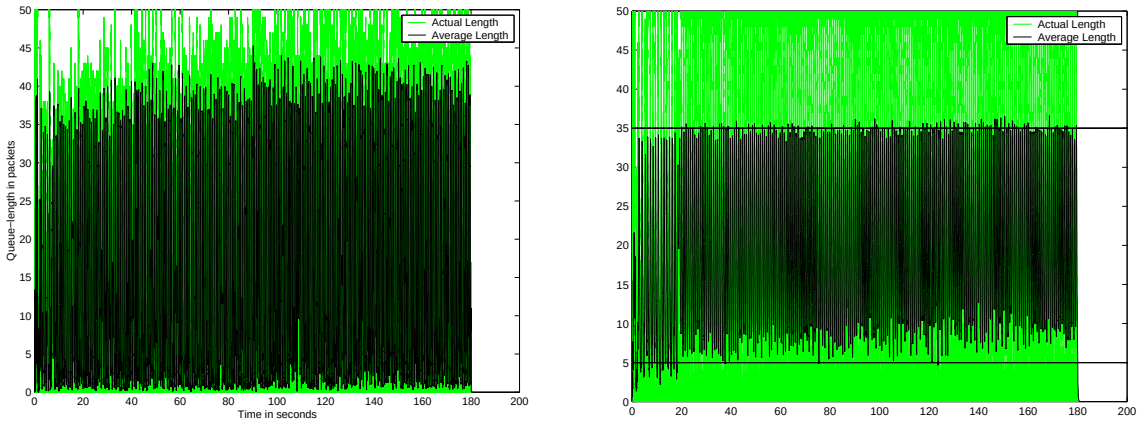


(a) (b)

Figure 5.9: Queue-length variations: (a) RED case; (b) Fuzzy RED case.

Figure 5.9 shows that although the overall long run average queue-length is quite similar for RED and Fuzzy RED (around 25 packets), the variation in queue length of RED is significantly higher with RED than with Fuzzy RED. High variation in queue-length results in high delay variation (jitter), thus decreasing the quality of voice services.

Scenario 4. *Queue-length variation with different RTTs.* To simulate a high level of variation of incoming TCP traffic, we set access link delays to range from 100 ms to $(100 + N - 1)$ ms, where N is the number of connections (nodes).



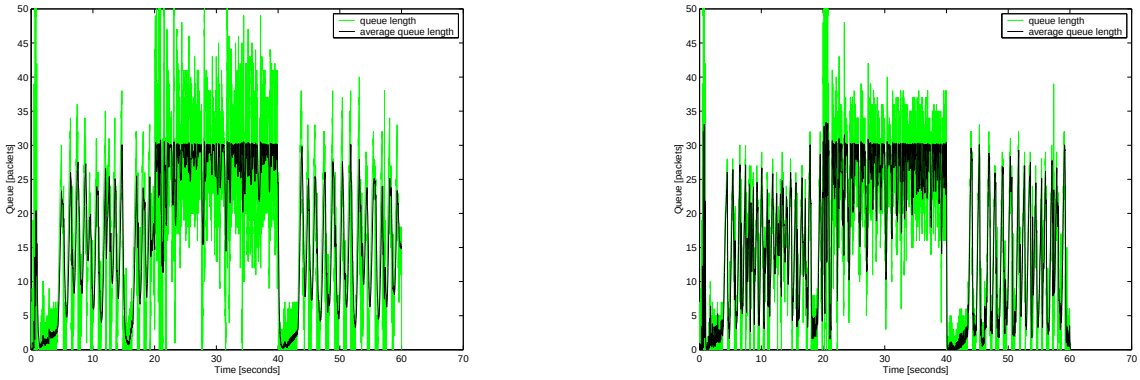
(a) (b)

Figure 5.10: Queue-length variations: (a) RED case; (b) Fuzzy RED case.

As we can see in Figure 5.10, queue-length variation is significantly higher with RED than with Fuzzy RED. Moreover, Figure 5.9 and Figure 5.10 clearly show that performance of both RED and Fuzzy RED, respectively, decreases when we simulate with different RTTs.

5.6.2 Performance with Non-Stationarities

We examine the performance of Fuzzy RED with level-shifts, which are the most common non-stationarity effects observed. We run the simulation in three parts each with length of 20 seconds. First, 10 TCP flows are active. After 20 seconds, an addition of 10 TCP flows enter. After 20 seconds, these 10 flows are terminated. All other parameters are the same as the simulation for the stationary case.

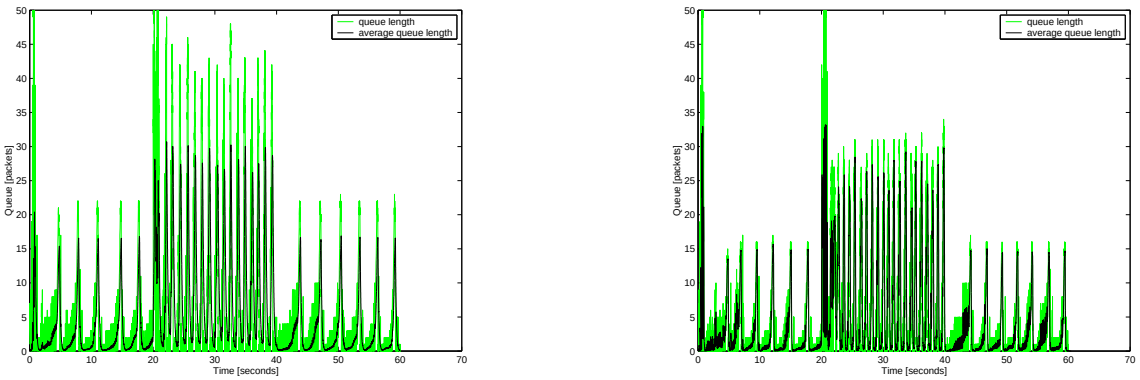


(a) (b)

Figure 5.11: Performance with level-shifts: RED vs. Fuzzy RED: (a) RED case; (b) Fuzzy RED case.

Figure 5.11 shows the dynamic of queue-length with RED and Fuzzy RED. After the increase in workload (additional 10 flows enter), actual queue-length with RED varies widely in the full range between 1 and 50. Fuzzy RED adapts to the sudden change in condition, and does not allow the queue-length to change quickly, keeping the actual queue-length in the target of 15 to 35 packets. After the decrease in workload (10 flows leave), it takes around 2 seconds for both RED and Fuzzy RED to get back to a normal condition, but Fuzzy RED produces somewhat smaller values for average queue-length and queue-length variation.

Scenario 6. *Performance with background traffic.* We run the simulation with some background web (http) traffic by adding short http sources to the examined long FTP connections. Each http source sends a request (a packet) to its destination, which replies with a file of size that is exponentially distributed with a mean of 125 KB-packets (the file size distribution can also be modelled by the Weibullian distribution, but here, we use exponential distribution, for the sake of simplicity). The waiting time for another request is also exponentially distributed with a mean of 1 second.



(a) (b)

Figure 5.12: Performance with Non-Stationarities: RED vs. Fuzzy RED: (a) RED case; (b) Fuzzy RED case.

We observe in Figure 5.12 periodicity with both RED and Fuzzy RED. Periodicity is well-known in the dynamics of queue-length in a buffer and it seems that Fuzzy RED does not filter out periodicity, but it adapts to changes somewhat quicker. More importantly, although the average queue-lengths both for Default Original RED and Fuzzy RED are similar, the actual queue-length is higher with RED than with Fuzzy RED.

5.7 Conclusion

In this chapter, a comprehensive performance analysis of the RED queue management scheme was presented. We demonstrated that RED in general does not guarantee proportional loss to flows and gave a proof for the TCP case. We also analytically evaluated the

performance of the EWMA algorithm in RED.

We found that the EWMA algorithm in RED is an unbiased estimator of average queue-length, regardless of the weighting value w_q. We also pointed out the theoretical and practical limits of the EWMA in RED.

We proposed the use of Fuzzy EWMA to RED (Fuzzy RED). Simulation results show that our proposed Fuzzy RED improves RED performance in a number of router-based metrics such as packet loss rate, average queueing delay, link utilization, and global power.

Regarding future work in this direction, the application of *linear adaptive predictor* to predict the queue length in RED is a possible and promising approach.

Chapter 6

Summary of the Dissertation

The objective of this dissertation was to provide a comprehensive performance analysis of some key traffic control techniques in TCP/IP network.

Motivation and previous work were first briefly overviewed in Chapter 1. The finding and results of the dissertation were presented in detail in the next four chapters. In Chapter 2 new metrics as well as novel algorithms to characterize the dynamics of TCP traffic were presented. In Chapter 3 a new unified model for TCP was provided. Game-theoretic analysis of rate control and the parameter setting of TCP Vegas was presented in Chapter 4. In Chapter 5, a comprehensive performance analysis of the Random Early Detection scheme was provided.

In what follows, we summarize the main contributions of our work.

6.1 Measurement of the Metrics of TCP

In Chapter 2 the introduction, design and implementation of several new metrics and novel algorithms to measure different aspects of TCP traffic were presented. These include:

- The measurement of the number of forward-going packets of TCP connections.
- The detection of the states of TCP.
- The measurement of state-based metrics of TCP.

Regarding the measurement of the number of forward-going packets of TCP connection, we have introduced a new concept, namely the virtual queue. We have discussed the relationship of the virtual queue concept with other important metrics of TCP such as the congestion window as well as the number of out-going packets of a TCP connection. We have implemented the proposed algorithm and carried out simulation as well as measurement analysis to validate it. We have also shown how to use this metric to have a better insight into the dynamics of TCP.

Regarding the detection of the states of TCP, we have introduced, designed and implemented a novel algorithm to detect the state changes of TCP during a connection. We have carried out simulation analysis to validate the proposed algorithm. We have also shown the applicability of this algorithm to the analysis of TCP dynamics as well as to the validation of the results (models) presented in this dissertation.

Being able to detect the state changes of TCP, we have introduced, designed and implemented a number of new algorithms to measure of the state-based metrics of TCP. These include:

- The sojourn time distribution at each state during a TCP connection.
- The jumping probabilities from one state to another state during a connection.
- The distribution of the number of packets sent in each time slot (RTT).

We have carried out simulation analysis to validate the proposed algorithms. We have also shown how to use the proposed algorithm to collect the state-based metric to validate our state-based TCP model presented in this dissertation.

The results presented in Chapter 2 are published in [J4, C7, C8].

6.2 New Unified Model for TCP

In Chapter 3, a novel approach to model TCP traffic was presented. One of the benefits of our approach is that we can build a unified model for different versions of:

- We have proposed a new unified model for different versions of TCP based on the states of TCP itself. We considered the dynamics of TCP as a Discrete-time Batch Markovian Arrival Process (D-BMAP), where the states of TCP are the states of the modulating Markov chain of this process.

Another benefit of our approach is that it provides us a simple way to characterize a TCP connection:

- We have introduced a new concept, namely the TCP characterization matrix, to characterize a TCP connection. We have also shown how to compute the elements of the introduced TCP characterization matrix.

We have carried out simulation to validate the new model in different scenarios, with different versions of TCP.

The results presented in Chapter 3 are published in [J1, J3, C2, C4].

6.3 Game-Theoretic Analysis of TCP Vegas

In Chapter 4, a game-theoretic analysis of TCP Vegas was presented. We have investigated the rate control and parameter setting problems of TCP Vegas from a game-theoretic point of view.

Regarding the rate control problem of the all-TCP Vegas network, we have found that there exists a unique Nash equilibrium (in its pure strategy space) for the TCP Vegas rate control game.

Regarding the parameter setting problem of all-TCP Vegas network, we have studied different games under different scenarios and found that under the selfish behaviour of the end-users the resulted Nash equilibria (if at all exist) are very inefficient and the network is prone to congestion.

We have also extended the analysis to FAST TCP case and concluded that under the selfish behaviour of the end-user, the network is prone to congestion. This poses a serious threat to the possible deployment of FAST TCP in the future Internet.

The results presented in Chapter 4 are published in [C0, C1].

6.4 Performance Analysis of RED

In Chapter 5, a comprehensive performance analysis of RED was presented. The main results of the chapter can be summarized by the followings:

- We have pointed out that RED, in general, does not possess proportional loss between flows as claimed and widely adopted in previous research. We have suggested the generalization of the PASTA property and give a proof for TCP flows.
- We have evaluated the performance of the Exponential Weighted Moving Average (EWMA) algorithm in RED. We have found that EWMA in RED is an unbiased estimator of the average queue-length, regardless of the weighting value w_q. We have pointed out the theoretical and practical limits of EWMA in RED.
- We have proposed the use of Fuzzy EWMA to RED (Fuzzy RED) to alleviate the inflexibility of RED tuning.
- We have carried out a comparative simulation-based analysis of different versions of RED and shown that in the case of a high workload and a high level of variation, Fuzzy RED, by tracking system variation in an on-line manner, improves RED performance in a number of important router-based metrics like packet loss rate, average queueing delay, link utilization, and global power.

The results presented in Chapter 5 are published in [J2, C5].

Appendix A

Appendices of Chapter 2

A.1 State detection in more detail

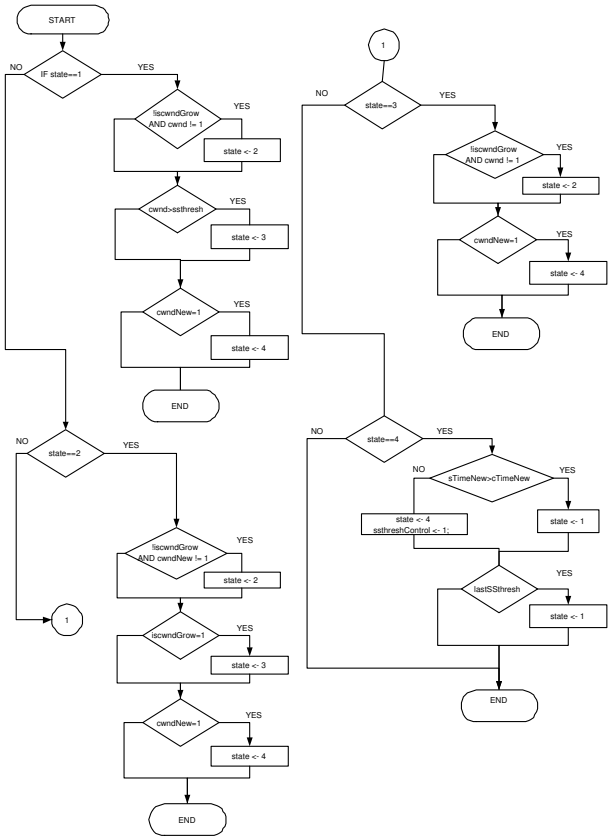


Figure A.1: State detection flow diagram

Bibliography

[Ake02] A. Akella, R. Karp, C. Papadimitrou, S. Seshan, and S. Schenker, *Selfish behavior and stability of the Internet: A game-theoretic analysis of TCP* ACM SIGCOMM, 2002.

[AlBa04] T. Alpcan, T. Basar, *Distributed algorithms for Nash equilibira of flow control games,* to appear in Annals of Dynamic Games, 2004

[ALLY01] S. Athuraliya, V. Li, S. Low, Q. Yin, *REM: active queue management,* IEEE Network, June 2001.

[AOMC01] J. Aweya, M. Ouellette, D. Montuno, A. Chapman *A Control Theoretic Approach to Active Queue Management* Computer Networks, 36, 2001.

[BloCa95] C. Blondia, O. Casals, *Statistical multiplexing of VBR sources: A matrix-analytic approach,* Performance Evaluation 16, pp. 5-20, 1992.

[BMP94] L. Brakmo, S. O'Malley, and L. Peterson, *TCP Vegas: new techniques for congestion detection and avoidance,* IEEE/ACM SIGCOMM 94, London, UK, Sept. 1994.

[Bon98] T. Bonald, *Comparison of TCP Reno and TCP Vegas,* Workshop on the modeling of TCP, 1998.

[Bur76] P. Burke, *Proof of a Conjecture on the Interarival-Time Distribution in M/M/1 Queue with Feedback,* IEEE Trans. on Communication, vol. 24, 1976.

[CaMe00] C. Casetti and M. Meo, *A new approach to model the stationary behavior of TCP connections.* In Proc. of IEEE INFOCOM, pages 367–375, March 2000.

[CJOS00] M. Christiansen, K Jeffay, D. Ott, F. D. Smith, *Tuning RED for Web traffic* ACM SIGCOMM'00, Stockhom, 2000.

[FGS01] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. *Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management,* ACIRI Technical Report, 2001.

[FKSS99] W. Feng, D. Kandlur, D. Saha, and K. G. Shin. *BLUE: A New Class of Active Queue Managemnent Algorithms,* UM CSE-TR-387-99, April 99.

[Flo03] Sally Floyd, *HighSpeed TCP for Large Congestion Window,* RFC 3649, December 2003.

[FloJa93] Sally Floyd and Van Jacobson *Random Early Detection Gateways for Congestion Avoidance,* IEEE/ACM Transactions on Networking, vol. 1, no. 4, August 1993, pp. 397-413.

[HBT99] P. Hurley, J. Boudec, and P. Thiran, *A Note on the Fairness of Addictive Increase and Multiplicative Decrease,* ITC 16, UK, 1999.

[HeLu86] H. Heffes, D. Lucantoni, *A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performace,* IEEE Journal on Selected Areas in Communications, Vol. 4, No. 6, September 1986, pp. 856-867.

[HMTG01] C. V. Hollot, V. Misra, D. Towsley and W. Gong, *A Control Theoretic Analysis of RED,* INFOCOM 2001, Alaska, April 22-26, 2001.

[Jac88] V. Jacobson, *Congestion avoidance and control,* Proceedings of ACM SIGCOMM'88, August 1988.

[JWL04] Cheng Jin, David X. Wei and Steven H. Low, *FAST TCP: motivation, architecture, algorithms, performance,* IEEE INFOCOMM'04, March 2004.

[KaPa87] Karn, P. and C. Partridge, *Improving Round-Trip Time Estimates in Reliable Transport Protocols,* SIGCOMM 87.

[Kel03] Tom Kelly, *Scalable TCP: Improving Performance in Highspeed Wide Area Networks,* ACM SIGCOMM Computer Communication Review, Vol. 33, Issue 2, pp. 83-91, April 2003.

[Kes91] S. Keshav, *A Control-theoretic Approach to Flow Control,* Proc. ACM SIGCOMM 1991, Sept. 1991

[Kle75] L. Kleinrock, *Queueing Systems: Theory* John Wiley and Sons, 1975.

[Kor95] Y. A. Korilis and A. A. Lazar, *On the existence of equilibria in noncooperative optimal flow control,* Journal of the ACM, vol. 42, no 3 pp. 584-613, 1995.

[Kum98] A. Kumar, *Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link,* IEEE/ACM Transactions on Networking, 1998.

[LiMo97] D. Lin and R. Morris, *Dynamics of Random Early Detection,* SIGCOMM'97.

[LoLa99] S. Low and D. Lapsley, *Optimization flow control, I: basic algorithm and convergence,* IEEE/ACM Transactions on Networking, 7(6):861-874, December 1999.

[Low03] S. Low, *A duality model of TCP and queue management algorithms,* IEEE/ACM Transactions on Networking, October 2003.

[LPW02] S. Low, L. Peterson, and L. Wang, *Understanding Vegas: a duality model,* Journal of ACM, 49(2):207-235, March 2002.

[LTWW93] W. Leland, M. Taqqu, W. Willinger, D. Wilson, *On the Self Similar Nature of Ethernet Traffic,* ACM SIGCOMM 93, San Francisco, CA, USA, Sept. 1993.

[MaVa94] J. MacKie-Mason, H. Varian, *Pricing the Internet,* in B. Kahin and J. Keller, eds., Public Access to the Internet, MIT Press, Cambridge, MA, 1995.

[MBB00] M. May, T. Bonald, and J. Bolot. *Analytic Evaluation of RED Performance,* Proc. of INFOCOM'00, 2000.

[MeYa95] B. Melamed and D. Yao, *The ASTA Property,* Frontiers in Queuing: Models, Methods, and Problems, CRC Press, 1995.

[MGT00] V. Misra, W. Gong and D. Towsley, *A Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED,* SIGCOMM 2000.

[MMFR96] M. Matthis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgement Options,* October 1996, RFC 2018.

[MSMO97] M. Matthis, J. Semske, J. Mahdavi, and T. Ott, *The Macroscopic Behavior of the TCP Congestion Avoidance Mechanism,* Computer Communication Review, 27(3), July 1997.

[Nag84] J. Nagle, *Congestion Control in IP/TCP,* RFC 896, January 1984.

[Neu81] Marcel Neuts, *Matrix-Geometric Solutions in Stochastic Models - An Algorithmic Approach,* The Johns Hopkins University Press, Baltimore, Maryland, 1981.

[NS2] NS2 software and documentation are available at the following site: http://www.isi.edu/nsnam/ns/

[OKM96] T. Ott, J.H.B. Kemperman, M. Mathis, *The Stationary Behavior of Ideal TCP Congestion Avoidance,* Bell Lab Technical Report, 1996.

[OsRu94] M. J. Osborne and A. Rubenstein, *A course in game theory,* Cambridge, Massachusetts: The MIT Press, 1994.

[Pax94] V. Paxson, *Growth Trends in Wide-Area TCP Connections,* IEEE Network, 8(4), pp. 8-17, July/August 1994.

[Pax97] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics,* Ph.D. Dissertation, University of California, Berkeley, CA, April 1997.

[PFTK98] J. Padhye et al, *Modeling TCP Reno Throughput: A Simple Model and Its Empirical Validation,* SIGCOMM'98, 1998.

[RFC2001] Richard Stevens, *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,* January, 1997, RFC 2001.

[RFC2525] V. Paxson, M. Allman, S. Dawson, I. Heavens, B. Volz, *Known TCP implementation problems,* RFC 2525, March 1999.

[RFC2582] S. Floyd, T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm,* RFC 2582.

[RFC2988] V. Paxson, M. Allman, *RFC 2988* http://rfc.net/rfc2988.html

[Ros65] J. B. Rosen, *Existence and uniqueness of equilibrium points for concave n-person games,* Econometrica, vol. 33, pp. 520-534, Jul. 1965.

[Sch95] Scott Schenker, *Making Greed Work in Networks: A game-theoretic analysis of switch service disciplines,* IEEE/ACM Transactions on Networking, vol. 3, 1995.

[Sch97] Misha Schwarz, *Telecommunication Networks: Protocols, Modelling and Analysis* Addision Welsley, 1997.

[VaFe01] G. Vattay, A. Fekete, *Self-Similarity in Bottleneck Buffers,* in Proc. of Globecom 2001.

[VeBo00] A. Veres, M. Boda, *The Chaotic Nature of TCP Congestion Control,* INFOCOM 2000, Tel Aviv, 2000.

[VMKV00] A. Veres, S. Molnár, Zs. Kenesei, G. Vattay, *On the Propagation of Long Range Dependence in the Internet,* ACM SIGCOMM 2000, Stockholm, Sweden, August 28 - September 1, 2000.

[Wol82] R. Wolff, *Poisson Arrivals See Time Averages* Operations Research, vol. 30, no 2, 1982.

[WyZu02] B. Wydrowski, M. Zukerman, *GREEN: An Active Queue Manangement Algorithm for a Self Managed Internet,* in Proc. of ICC 2002, New York, vol. 4, pp. 2368-2372, 2002.

[YMKT99] M. Yajnik, S. Moon, J. Kurose and D. Towsley, *Measurement and Modelling of the Temporal Dependence in Packet Loss,* in Proc. of INFOCOM 1999.

[ZCR00] M. Zorzi, A. Chockalingam, and R. Rao *Throughput analysis of TCP on channels with memory,* IEEE Journal on Selected Areas in Communications, vol. 18, no. 7, pp. 1289–1300, 2000.

[ZhAt00] B. Zheng, M Atiquizzaman, *DSRED: Improving Performance of Active Queue Management over Heterogenous Networks,* The 25th Annual IEEE Conference on Local Computer Networks, November 9-10, Tampa, Florida, USA, 2000.

Publications

Journal papers

[J0] S. Molnár, **T. A. Trinh**. Congestion Games in TCP Vegas and Their Applications in FAST TCP. Submitted to *Telecommunications Systems*, 2004.

[J1] **T. A. Trinh**, S. Molnár. Modeling and Analysis of TCP Traffic: A State-based Approach. *under submission*, 2004.

[J2] **T. A. Trinh**, S. Molnár. A Comprehensive Performance Analysis of Random Early Detection Mechanism. *Telecommunications Systems,* 25(1-2): 9-31, January-February, 2004.

[J3] **T. A. Trinh**, S. Molnár. Modeling TCP Traffic: A State-based Approach. *Periodica Polytechnica, Electrical Engineering,* Vol. 48, No. 1, pp. 1-14, 2004.

[J4] **T. A. Trinh**, T. Éltető. On the Stability of TCP. *Journal on Communications,* November-December 2000.

Conference papers

[C0] **T. A. Trinh**, S. Molnár. Understanding TCP Vegas and FAST TCP: A Game-Theoretic Perspective, submitted to *IEEE/IFIP Networking 2005.*

[C1] **T. A. Trinh**, S. Molnár. A Game-Theoretic Analysis of TCP Vegas. In *Proc. of QofIS'04 - Quality of Service in the Emerging Networking Panorama*, **Springer** *Lecture Notes in Computer Science 3266 (LNCS 3266)*, pp. 338-347, Barcelona, Spain, September 29 - October 1, 2004.

[C2] **T. A. Trinh**, S. Molnár. A Novel Approach to Model TCP Traffic, in *Proc. of IEEE GLOBECOM 2004,* Dallas, Texas, USA, November-December, 2004.

[C3] **T. A. Trinh**, B. Sonkoly, S. Molnár. A Study of HighSpeed TCP: Observations and Re-evaluation, in *Proc. of EUNICE 2004,* Tampere, Finland, June 2004.

[C4] **T. A. Trinh**, S. Molnár. A State-based Analysis of TCP, in *Proc. of IFIP Workshop on Next Generation Networks,* Hungary, 8-10 September 2003.

[C5] **T. A. Trinh**, S. Molnár. RED Revisited, in *Proc. of The 10th International Conference on Telecommunication Systems Modeling and Analysis,* CA, USA, October 3-6, 2002.

[C6] **T. A. Trinh.** On the Estimation of Average Queue-length in RED, in *Proc. of PCH Conference on Telecommunications,* Budapest, Hungary, April 2001.

[C7] **T. A. Trinh**, T. Éltető, L. Györfi. On Some Metrics of TCP, In *Proc. of The 25th International Conference on Local Area Networks*, Florida, USA, November 2000.

[C8] **T. A. Trinh.** On the Stability of TCP, in *Proc. of Students Scientific Conference,* Budapest Univ. of Technology and Economics, Budapest, Hungary, November 1999.