End-to-end congestion control over wireless networks : analysis and enhancements

Zhou, Bin

2008

Zhou, B. (2008). End-to-end congestion control over wireless networks : analysis and enhancements. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/13605>

<https://doi.org/10.32657/10356/13605>

**END-TO-END CONGESTION CONTROL OVER
WIRELESS NETWORKS:
ANALYSIS AND ENHANCEMENTS**

Bin ZHOU

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirements for the degree of
Doctor of Philosophy

2008

Acknowledgments

First and foremost, I would like to express the deepest gratitude to my supervisor, Professor Fu Cheng Peng, for sharing his passion for research with me and being such a great advisor for both academic and personal matters. His perfect writing skills, his creative idea and quick thinking impress me deeply. I have greatly benefited both from his excellent technical advice in research work and from his useful suggestions in daily experience. I would also like to thank Professor Ngoh Lek Heng, my co-supervisor, for showing me another interesting research area about sensor networks in my early PhD study.

I would like to thank my group research partners Yu Kuan, Zhang Ke, Zhang Chuenlei, and Lv Qian. I benefited greatly from their invaluable suggestion and help in my research work. I would also like to thank my lab colleagues and technicians, for their self-giving help and warmhearted assistant for research facilities and experiments.

Special thanks to Professor Kwan Lawrence Yeung at The University of Hong Kong, and his students Tony Kwan and Huang Wen, for their kind help on conducting the Internet experiments in my thesis. I am also indebted to Professor Victor O. K. Li, for his invaluable comments on my research papers.

This thesis is dedicated to my parents and my wife, for their constant love, encourage, and support in my life. They share with me the happiness and the sorrow, the success and the frustration. Words cannot express my gratitude to them, and I do not know how to pay back my debt to them for their sacrifice. I will never know.

Contents

Acknowledgments	i
List of Figures	v
List of Tables	viii
Abstract	ix
1 Introduction	1
1.1 Research Background	1
1.2 Problem Statement and Motivation	3
1.2.1 Wireless Issues in End-to-end Congestion Controls for Bulk Data Transfer	3
1.2.2 Wireless Issues in End-to-end Congestion Controls for Streaming .	4
1.3 Summary of Contributions	5
1.3.1 Analysis and Enhancements of End-to-end Congestion Controls for Bulk Data Transfer over Wireless Networks	6
1.3.2 Analysis and Enhancements of End-to-end Congestion Controls for Streaming over Wireless Networks	7
1.4 Outline of The Thesis	8
2 Literature Review	10
2.1 End-to-end Congestion Controls for Bulk Data Transfer	10
2.1.1 TCP Reno	10
2.1.2 TCP Veno	19
2.1.3 Other TCP Variants over Wireless Networks	22
2.2 End-to-end Congestion Controls for Streaming	26
2.2.1 TFRC	27

2.2.2	MULTFRC	30
2.2.3	Other TFRC Variants over Wireless Networks	31
3	Analysis and Enhancements of End-to-end Congestion Controls for Bulk Data Transfer over Wireless Networks	34
3.1	Throughput Model for TCP Veno	34
3.1.1	Veno Equation	35
3.1.2	Validation of Veno Equation	43
3.1.3	Summary	49
3.2	TCP Veno Enhancement over Light-load Wireless Networks	50
3.2.1	TCP Veno With <i>Congestion Loss Rate</i>	50
3.2.2	Performance Evaluation	54
3.2.3	Summary	60
3.3	TCP Veno Enhancement over High Bandwidth-Delay Product Networks	60
3.3.1	TCP Veno with $pBDP$	60
3.3.2	Performance Evaluation	67
3.3.3	Summary	73
4	Analysis and Enhancements of End-to-end Congestion Controls for Streaming over Wireless Networks	74
4.1	Veno Equation Based TFRC Enhancement	74
4.1.1	TFRC Veno	75
4.1.2	Performance Evaluation	76
4.1.3	Summary	86
4.2	Veno State Differentiator Based TFRC Enhancement	86
4.2.1	TFRC with Veno State Differentiator	87
4.2.2	Performance Evaluation	88
4.2.3	Summary	93
4.3	TFRC Veno Implementation	94
4.3.1	DCCP in Linux Kernel	95
4.3.2	TFRC Veno Module	101
4.3.3	Summary	104

5 Conclusion and Future Work	105
5.1 Summary	105
5.2 Future Work	108
References	112
Publications	122

List of Figures

2.1	Four key algorithms of TCP Reno.	11
2.2	Slow start.	12
2.3	Operations of acknowledgement in TCP receiver.	14
2.4	Overview of TFRC mechanisms.	27
2.5	Estimation of loss events.	29
3.1	Congestion window evolutions of Veno and Reno.	35
3.2	Packets sent during a TD period.	37
3.3	Topology of NS-2 simulation.	43
3.4	Simulation results under exponential random loss: single flow.	44
3.5	Simulation results under two-state random loss: single flow.	45
3.6	Simulation results under exponential random loss: background traffic.	45
3.7	Simulation results under two-state random loss: background traffic.	46
3.8	More complex topology of NS-2 simulation.	46
3.9	Simulation results on a complicated topology.	47
3.10	Topology of the Internet experiments.	48
3.11	The Internet experimental results.	49
3.12	Topology in NS-2 experiments.	51
3.13	Window evolution of Veno-LL and Veno, packet loss rate = 0.01, bottle- neck bandwidth = 8Mbps.	52
3.14	Calculation of congestion loss rate.	53
3.15	Throughput improvement of Veno-LL under different number of connec- tions (N) and different random loss rates (Pr).	55
3.16	Fairness comparison between Veno-LL and Veno.	56

3.17	Friendliness comparison between Veno-LL and Veno.	57
3.18	Topology of WLAN experiments.	58
3.19	Experiments over WLAN: light load case.	59
3.20	Experiments over WLAN: heavy load case.	59
3.21	Topology in NS-2 experiments.	62
3.22	Random loss impacts on Veno in three BDP cases.	63
3.23	Congestion window evolutions of TCP Veno.	63
3.24	Estimation of $\mu\tau$ under different random loss rates.	66
3.25	Congestion window evolutions of Veno-HB and Veno over 2Mbps, 300ms bottleneck link.	68
3.26	Throughput improvement of Veno-HB under different BDP cases and dif- ferent random loss rates (Pr).	68
3.27	Throughput improvement of Veno-HB under multiple connections (N) and different random loss rates (Pr).	69
3.28	Fairness comparison between Veno-HB and Veno.	70
3.29	Friendliness comparison between Veno-HB and Veno.	71
3.30	Sequence numbers against time over a 2Mbps, 40ms link.	72
3.31	Sequence numbers against time over a 2Mbps, 240ms link.	72
3.32	Sequence numbers against time over a 12Mbps, 40ms link.	72
4.1	Topology of simulation experiments.	77
4.2	Throughput comparisons under different random loss rates.	78
4.3	Throughput comparisons under multiple connections (packet loss rate is 0.01).	78
4.4	Sequence number against time, 16 connections, random losses follow ex- ponential distribution, packet loss rate is 0.01.	79
4.5	Fairness comparisons under different random loss rates.	80
4.6	Fairness comparisons under multiple connections (packet loss rate is 0.01).	81
4.7	TCP-friendliness comparisons under different random loss rates.	81
4.8	TCP-friendliness comparisons under multiple connections (packet loss rate is 0.01).	82

4.9	Sending rate variation comparisons under different random loss rates. . .	83
4.10	Sending rate variation comparisons under multiple connections (packet loss rate is 0.01).	84
4.11	Topology of the Internet experiments.	85
4.12	Topology of NS-2 experiments.	89
4.13	Throughput comparison.	90
4.14	Receiving ratio comparison.	91
4.15	CoV comparison.	92
4.16	Fairness of Enhancement with different values of β	93
4.17	Friendliness of Enhancement with different values of β	94
4.18	DCCP functions.	97
4.19	CCID pointer array.	98
4.20	TFRC/TFRC Veno module.	99
4.21	Table for the values of $f(p)$	101
4.22	Table for the values of $c1(\gamma)$ and $c2(\gamma)$	103

List of Tables

3.1	Throughputs of Veno over 2Mbps and 8Mbps.	51
4.1	The Internet experiment results on single flow.	85
4.2	The Internet experiment results on coexisting flows.	85
5.1	Comparisons of TFRC, MULTFRC, and TFRC Veno.	107

Abstract

The success of the Internet can be attributed to the large number of useful applications running on it. Basically, these applications can be classified into two categories – bulk data transfer and streaming transfer – based on the different requirements on the data transmission. As we know, bulk data transfer applications, such as file transfers and emails, usually require high reliability of data transmissions while delays or delay variations caused are less concerned. On the other hand, streaming applications, including real-time audio and video, are very sensitive to delays and delay variations, while certain rate of packet losses is acceptable.

In order to provide the end-to-end congestion control for these two different types of applications, two end-to-end congestion control algorithms, namely TCP and TFRC, have been carefully designed over last years. To date, both algorithms work very well in wired networks. However, in wireless networks where random loss is rampant due to environmental noise, both TCP and TFRC suffer significant performance degradation unnecessarily, because they often misinterpret the random loss as the indication of network congestion.

This thesis studies such wireless issue for both TCP and TFRC. The analysis of TCP focuses on a novel wireless TCP variant, TCP Veno. Specifically, we develop a throughput model for TCP Veno, to provide theoretical analysis for TCP Veno evolution over wireless networks. The accordingly derived equation, called the “Veno equation”, characterizes TCP Veno congestion control algorithm in both wired and wireless situations. Our extensive simulations and the Internet experiments demonstrate the accuracy of such throughput prediction. At the same time, we also enhance TCP Veno in different ways, after observing that Veno behaves conservatively in certain wireless environments.

Very importantly, the analysis of TCP Veno provides us the clues to further improve TFRC performance over wireless networks. Based on the Veno equation obtained, we

innovatively propose an enhancement of TFRC called TFRC Veno. TFRC Veno intelligently makes use of the Veno equation to calculate the sending rate of the streaming data, instead of the original Reno equation. Simulations and the Internet experiments have proved that, TFRC Veno has up to 300% throughput improvement over the original TFRC in wireless environments. Meanwhile, such improvement does not cause any bias to the existing TFRC or other TCP-friendliness connections. The salient feature is that TFRC Veno only requires a simple modification at the sender side of TFRC and there is no change at the receiver protocol stack or intervention at any of the intermediate network nodes. Therefore, we can say such enhancement can be easily deployed in the current Internet.

At last, with the help of Veno state differentiator, the thesis also explores how to enhance TFRC performance over wireless networks in another way.

Chapter 1

Introduction

1.1 Research Background

The Internet is a collection of interconnected networks that offers a best-effort data transmission service to users using the Internet Protocol (IP) [1]. The IP protocol provides the means to connect diversity of networks such as Ethernet, TokenRing, FDDI, SONET, HDLC and ATM. The advantages of IP connectionless design, flexibility and robustness have led a great success of the Internet over the past 20 years. However, the lack means to limit the packet sending rate in the IP protocol can also result in severe service degradation or “Internet meltdown”, when the users of the network collectively demand more resources of link bandwidth and buffer space than the network has to offer. This phenomenon was first observed during the early growth phase of the Internet of the mid 1980s [2], and is technically called “congestion collapse”. Congestion collapse motivated the use of end-to-end congestion control in the Internet to avoid such performance degradation [3].

The original end-to-end congestion control algorithm was provided by Van Jacobson [4] in 1988, and later turned into the dominating transport protocol in the Internet, TCP [5]. Based on the principle of additive-increase/multiplicative-decrease (AIMD), TCP probes for extra bandwidth by increasing its congestion window linearly with time; once it detects the network congestion signalled by packet loss, it reduces its window by a factor of two. Here congestion window is used in TCP to determine the number of

CHAPTER 1. INTRODUCTION

packets that can be outstanding at any time. Under certain assumptions of synchronized feedback, the AIMD control scheme has been shown to converge to a stable and fair operating point [6]. Because TCP only involves end hosts (sender and receiver) in its congestion control mechanisms, it is also called end-to-end congestion control.

TCP and its variants have enjoyed much success in the Internet as they can support all bulk data transfer applications such as file transfers, web services, emails, and interactive terminals. However, TCP is not well-suited for several emerging applications called streaming applications, such as real-time audio and video. This is because the reliability and ordering semantics in TCP increases end-to-end delays and delay variations. Furthermore, many of these streaming applications do not react well to the large and abrupt reductions in transmission rate on packet losses, which is caused by the TCP-style AIMD algorithms [7][8]. As a result, another kind of end-to-end congestion control mechanisms, which targets the low end-to-end delay, small delay variation, and smooth sending rate, was proposed to support these streaming communications. TCP-Friendly Rate Control (TFRC) [9][10] is one of the most popular protocols. It is equipped with an equation that can model the throughput of TCP [11] in the same network situations. During transmission, TFRC adjusts its sending rate according to the calculated result from the equation, making its instant throughput similar to that of TCP under the same circumstances. Thus TFRC does not bias any other existing TCP-like traffics during transmission.

In 2000, S. Floyd proposed the principles of designing end-to-end congestion control [12], and pointed out three goals of end-to-end congestion control: 1) Preventing congestion collapse. This is the initial reason leading to end-to-end congestion control, and has been elaborated in the previous part of this section. 2) Fairness. Flows belonging to the same end-to-end congestion control should share the network sources (i.e., bandwidth or buffer space) fairly. Furthermore, new end-to-end congestion control algorithm should

not bias existing TCP-like traffics. The latter is also called TCP-friendliness. 3) Optimizing performance regarding throughput, delay, and loss. That is to say, end-to-end congestion control should not “overact” in the prevention of congestion collapse, which results in poor utilization on available network resources

1.2 Problem Statement and Motivation

End-to-end congestion controls for both bulk data transfer and streaming are originally designed for wired networks. However, wireless communication technology has been making significant progress in the recent past and will be playing a more and more important role in access networks, as evidenced by the widespread adoption of wireless local area network, wireless home networks, and cellular networks. Wireless networks have introduced new issues into end-to-end congestion controls.

1.2.1 Wireless Issues in End-to-end Congestion Controls for Bulk Data Transfer

It is known that wireless networks may suffer high packet loss rate. As described in Section 1.1, TCP always treats the occurrence of packet loss as a manifestation of network congestion. This assumption does not apply to networks with wireless channels, in which packet loss is often induced by noise, link error, or reasons other than network congestion. As in [13], we refer to such packet loss as random packet loss. Misinterpretation of random loss as an indication of network congestion causes TCP to back down the sending data rate unnecessarily, resulting in significant performance degradation [13][14][15].

Over the past decade, there have been many efforts to enhance TCP throughput by eliminating negative impacts from random losses. These enhancements can be divided into two categories: 1) Network-support enhancements, which need the support from the intermediate nodes in the network, such as routers, proxies, access points, or any other

kinds of devices. Snoop TCP [16], I-TCP [17], TCP with ECN [18], TCP with EBSN [19], and TCP with ELN [14] belong to this category. 2) End-to-end enhancements, which modified TCP mechanism only in the end nodes (sender or receiver or both). Typical examples are TCP Veno [20], TCP Westwood [21], TCP Jersey [22], TCP Santa Cruz [23], WTCP [24], TCP with FEC/ARQ [25], TCP-AP [26], C^3 TCP [27], and TCP-HO [28]. As compared to network-support enhancements, end-to-end enhancements do not need to add any new functions to the intermediate nodes in the network. This will avoid deployability problem in the real networks since it is costly and inconvenient to change the existing devices in the Internet.

As one of the end-to-end enhancements for TCP, TCP Veno provides efficient improvement in throughput by simply modifying the sender-side protocol stack. Many experiments have shown its superior performance, either in simulations or real networks [20][29]. TCP Veno is also incorporated by Linux Kernel since version 2.6.18 [30]. However, there is no theoretical analysis for TCP Veno to give an in-depth view on its behavior over wired and wireless networks. Moreover, whether its performance can be further improved is also an interesting issue.

1.2.2 Wireless Issues in End-to-end Congestion Controls for Streaming

Similar to TCP, TFRC also suffers severe performance degradation from wireless links. This is because TFRC is equipped with the equation modeling TCP throughput, and thus inherits the similar behaviors of TCP over wireless networks: TFRC interprets every packet loss as the indication of congestion and reduces its sending rate every time, resulting in unnecessary throughput degradation in wireless networks since a high percentage of losses are not caused by congestion.

Over recent years, some researchers have studied the throughput degradation problem of TFRC over wireless networks, and tried to eliminate such suffering from different ways.

These efforts can also be divided into two categories: 1) Network-support enhancements, which need the support from the intermediate nodes in the network. Typical examples are ECN-based TFRC [31], Proxy-based TFRC [32], WM-TFRC [33], and AED-based TFRC [34]. 2) End-to-end enhancements, which modified TCP mechanism only on the end nodes (sender or receiver or both). MULTFRC [35] belongs to this category. Similarly, end-to-end enhancements do not need to add any new functions to the intermediate nodes in the network as compared to network-support enhancements, and thus avoid deployability problem in the real networks.

While MULTFRC achieves much higher throughput than TFRC does over wireless networks, it suffers poor fairness and poor TCP-friendliness during transmission. This will be elaborated later in Chapter 4. How to enhance TFRC in throughput while not sacrifice other advantages of TFRC such as good fairness and good TCP-friendliness is still an ongoing issue.

1.3 Summary of Contributions

This thesis considers the wireless issues in end-to-end congestion controls, and provides analysis and enhancements for both bulk data transfer and streaming. The series of analysis and enhancements is based on TCP Veno. Specifically, we first derive a throughput model for TCP Veno. After the in-depth analysis on TCP Veno, we propose two enhancements to further improve its performance over wireless networks. Finally, we introduce the core idea of TCP Veno into TFRC, the end-to-end congestion control for streaming, and propose two enhancements for TFRC over wireless networks. Based on the difference of target applications, the contributions can be divided into two areas: works for bulk data transfer and works for streaming.

1.3.1 Analysis and Enhancements of End-to-end Congestion Controls for Bulk Data Transfer over Wireless Networks

- **Throughput Model for TCP Veno**

This work is the footstone of the thesis. It develops an analytic approach in packet level to characterize TCP Veno behavior in both wired and wireless situations. Being different from the equation of TCP Reno, a more general closed form equation called “Veno equation” is derived, taking into account of the dynamic congestion window reductions in Veno over wireless networks [36]. Extensive experiments, including simulations and live Internet tests, demonstrate that the Veno equation is able to accurately predict TCP Veno throughput over different network scenarios, ranging from very low lossy links to very heavy lossy links.

The throughput model provides a theoretical way to analyze TCP Veno behaviors during transmission. Moreover, the Veno equation can be used as a more advanced equation in TFRC to enhance its performance over wireless networks (seeing the following section).

- **TCP Veno Enhancement over Light-load Wireless Networks**

After the comprehensive analysis, TCP Veno is found to still miss much available bandwidth over wireless transmissions, especially when the network load is light and random losses are pervasive. A new variable, named *congestion loss rate*, is introduced in this work [37]. It helps Veno act more efficiently in response to random losses and grab more available bandwidth on the link, without any fairness or friendliness sacrificed. Simulations and real network experiments have shown the better performance of this enhancement as compared to the original Veno.

- **TCP Veno Enhancement over High Bandwidth-Delay Product Networks**

Another enhancement of TCP VenO aims to alleviate random loss impacts on TCP VenO over high Bandwidth-Delay Product (BDP) networks. This work is based on the observation that higher BDP of the network makes random loss cause more negative impact on TCP throughput [13]. An approximated variable called $pBDP$ is introduced to inform VenO situations of the network and refine its congestion window evolution [38]. Both simulations and real network experiments are used to verify its performance.

1.3.2 Analysis and Enhancements of End-to-end Congestion Controls for Streaming over Wireless Networks

- **Veno Equation Based TFRC Enhancement**

TFRC employs a throughput equation to control its sending rate on streaming. However, this equation is derived from TCP Reno (hereinafter, called the “Reno equation”), and thus unavoidably makes TFRC suffer severe performance degradation from wireless links. In this work, we explore to enhance TFRC over wireless networks using a more advanced equation – Veno equation – to replace the Reno equation. The new protocol is also referred as TFRC VenO [39].

Our extensive experiments including simulations and live Internet measurements demonstrate that TFRC VenO can achieve significant throughput improvement over TFRC while keep the same nice fairness, TCP-friendliness and smoothness as TFRC. Furthermore, TFRC VenO only involves modification of TFRC on the sender side without requiring any changes of the receiver protocol stack or intervention of the intermediate network nodes. As compared to another sender-modified enhancement MULTFRC, our proposal is much more friendly to other network traffic when achieving the same throughput improvement.

- **Veno State Differentiator Based TFRC Enhancement**

We also propose another way to introduce Veno idea into TFRC mechanism [40]. Specifically, it utilizes Veno’s state differentiator to distinguish congestion losses and random losses during transmission. By discounting the impact of those non-congestion losses on the throughput calculation, it can effectively alleviate the throughput degradation caused by wireless links. The simulation results have shown that, our proposal can achieve better throughput as compared to the original TFRC. Meanwhile, it maintains other merits of the original TFRC, such as sending rate smoothness, fairness, and TCP-friendliness.

1.4 Outline of The Thesis

The rest of the thesis is organized as follows:

Chapter 2 reviews the two categories of end-to-end congestion controls: for bulk data transfer and for streaming. In the bulk data transfer part, we mainly introduce the primary version of TCP, TCP Reno, and one of its enhancements over wireless networks, TCP Veno. Other TCP variants for wireless networks are also briefed. In the streaming part, we describe the mechanisms of TFRC and one of its wireless variants, MULTFRC, in detail. Other wireless enhancements of TFRC are also introduced.

Chapter 3 presents the detailed work on end-to-end congestion controls for bulk data transfer. We first describe the derivation of the throughput model for TCP Veno, and then provide two enhancements for TCP Veno to further enhance its performance over two different types of wireless networks. Simulations and real network experiments results are also given in this chapter.

Chapter 4 describes our work on end-to-end congestion controls for streaming. Two proposals are provided, both based on the core idea of TCP Veno: one is equipped with the Veno equation, which can model TCP Veno throughput; another uses Veno’s state

CHAPTER 1. INTRODUCTION

differentiator. Simulations and real network experiments results for these two proposals are also shown in this chapter.

Chapter 5 concludes the thesis with a summary of the works. We also give suggestions for future work and possible directions.

Chapter 2

Literature Review

This chapter reviews the end-to-end congestion controls for bulk data transfer and for streaming respectively. In the bulk data transfer part, we mainly introduce the primary version of TCP, TCP Reno, and one of its enhancements over wireless networks, TCP Veno. Other TCP variants for wireless networks are also given. In the streaming part, we describe the mechanisms of TFRC and one of its wireless variants, MULTFRC, in detail. Other wireless enhancements of TFRC are introduced as well.

2.1 End-to-end Congestion Controls for Bulk Data Transfer

As the end-to-end congestion control for bulk data transfer, TCP and its variants have made great success in the Internet. This section gives an overview on TCP, including the primary version, TCP Reno, and some enhancements for wireless networks, e.g., TCP Veno.

2.1.1 TCP Reno

TCP Reno [41] is the primary version of TCP, which contains all necessary algorithms of TCP such as slow start (SS), additive-increase/multiplicative-decrease (AIMD), fast retransmit and fast recovery (FF), and retransmission timeout (TO). With reference

CHAPTER 2. LITERATURE REVIEW

to Figure 2.1, Reno starts with the slow start (SS) phase and its congestion window ($cwnd$) is incremented exponentially until $cwnd$ reaches a threshold called slow start threshold ($ssthresh$). After that, the congestion window increases linearly till a packet loss is experienced, at which point the window is halved. This phase is called additive-increase/multiplicative-decrease (AIMD) phase. In order to retransmit the lost packet efficiently, Reno enters fast transmission and fast recovery (FF) phase until the packet is successfully received. However, sometime the sender is unable to know whether the packet is lost or not after a long time (e.g., network breakdown). In this case, Reno enters retransmission timeout (TO) phase to retransmit the timeout packets and restarts with the slow start phase. These four key algorithms will be described in detail in the following parts.

Figure 2.1: Four key algorithms of TCP Reno.

2.1.1.1 Slow Start (SS)

Slow start is used either at the start of a TCP connection or after a retransmission timeout (TO). The objective of slow start is to enable a TCP sender to discover the available bandwidth by increasing the amount of data injected into the network from initial window size of one segment, which prevents the TCP sender from congesting the network with a large burst of data.

CHAPTER 2. LITERATURE REVIEW

Figure 2.2: Slow start.

Figure 2.2 shows the procedure of the slow start. In TCP, every time a data packet arrives at the receiver side, the receiver responds with an acknowledgement (ACK). The sender starts by transmitting one segment and waits for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent. When both of these two segments are acknowledged, the congestion window is increased to four. This provides an exponential growth of congestion window. In particular, the pseudo-code of the slow start is shown in Listing 2.1.

Listing 2.1: Slow Start algorithm

```
//when an ACK arrives,
    cwnd = cwnd + 1;
```

By slow start Reno can quickly reach or approximate the equilibrium point of network system, where the capacity of the Internet is fully occupied. After reaching the initial *ssthresh* or getting an packet lost, Reno ends the slow start phase and comes into the AIMD phase.

The initial value of *ssthresh* is usually preset in Reno. However, many researchers have proposed to use estimated initial *ssthresh* to reduce or overcome the severe buffer overflow caused by the exponential growth of the congestion window [42][43][44][45]. M. Allman *et al.* also observed that the accuracy of sender-side estimated *ssthresh* is greatly limited, as compared to the estimation at receiver side, which results in a scalability problem while yielding better estimation than the sender-side estimation [46].

2.1.1.2 Additive-increase/multiplicative-decrease (AIMD)

Once slow start is completed, the network is expected having entered an equilibrium status, and then Reno uses the AIMD scheme to adjust its *cwnd*. The AIMD algorithm has two adjustment phases: 1) when the traffic load is steady or decreasing, the algorithm is in the probing phase. The internal traffic increases linearly to absorb released resources and probes the state of the bottleneck. 2) When a congestion signal (three duplicate ACKs, which will be discussed later) is received, the algorithm decreases the window size by half. This pattern of continually increasing and decreasing the congestion window continues throughout the lifetime of the connection.

As illustrated in Figure 2.1, the value of *cwnd* as a function of time in AIMD follows a “sawtooth” pattern. The important thing to understand about AIMD is that the source is willing to reduce its congestion window at a much faster rate than it is willing to increase its congestion window. The details of the AIMD algorithm are described in Listing 2.2.

Listing 2.2: Additive-increase/multiplicative-decrease algorithm

```
//when an new ACK arrives,
    cwnd = cwnd + 1/cwnd;
//otherwise a segment is lost,
    ssthresh = cwnd/2;
    cwnd = ssthresh;
```

Here the *ssthresh* is used as an estimator of system equilibrium point. Its choice is key to the performance of AIMD algorithm. Note that AIMD is sometime also called congestion avoidance (CA) in some literature.

2.1.1.3 Fast Retransmission and Fast Recovery (FF)

Before describing the FF algorithm, we take a closer look at the operations of acknowledgement (ACK) in TCP receiver. It is known that the receiver responds with a new ACK when a packet arrives. However, if the packet arrives out of order, the receiver

CHAPTER 2. LITERATURE REVIEW

will re-send the same ACK it sent last time. This second transmission of the same ACK is called a duplicate ACK. TCP generates duplicate ACKs (DUPACK) for every out-of-order segment received.

Figure 2.3: Operations of acknowledgement in TCP receiver.

As an example in Figure 2.3, the destination receives packets 1 and 2, but packet 3 is lost in the network, making packet 4, 5, and 6 out-of-order segments. Thus, the destination will send a duplicate ACK for packet 2 when packet 4 arrives, again when packet 5 arrives, and so on. When the sender sees the third duplicate ACK for packet 2, it notices the packet loss and retransmits packet 3. Note that when the retransmitted copy of packet 3 arrives at the destination, it then sends a cumulative ACK (ACK 6) for everything up to and including packet 6 back to the source.

The fast retransmission and fast recovery algorithms are usually implemented together. When three duplicate ACKs are received, the fast retransmission algorithm is triggered to retransmit the lost packet. After that, the fast recovery algorithm governs the transmission of new data until a non-duplicate ACK arrives. Specifically, 1) it sets

CHAPTER 2. LITERATURE REVIEW

ssthresh to one half of the current *cwnd*, and set *cwnd* to *ssthresh* plus 3. 2) Each time another duplicate ACK arrives, increase *cwnd* by one, and transmit a new segment if the *cwnd* is larger than the number of unacknowledged segments. 3) When the next ACK arrives that acknowledges new data, set *cwnd* to *ssthresh* (the value set in step 1). Listing 2.3 is the pseudo-code of these operations.

Listing 2.3: Fast retransmission and fast recovery algorithm

```

1)Retransmit the missing packet, and
   ssthresh = cwnd/2;
   cwnd = ssthresh + 3;
2)Each time another dup ACK arrives,
   cwnd = cwnd + 1;
3)When the next ACK acknowledging new data arrives,
   cwnd = ssthresh;

```

In some extent, FF can be seen as an extension of MD (multiplicative-decrease) algorithm since it just follows MD phase. After the end of FF phase, the AI (additive-increase) algorithm takes over starting from latest setting *ssthresh*.

2.1.1.4 Retransmission Timeout (TO)

Reno holds a variable *RTO* (Retransmit Time Out), which maintains the value of the time to wait for an ACK after sending a segment. If the waiting time exceeds *RTO*, Reno enters the retransmission timeout (TO) phase, which resets *cwnd* to 1 and enters the slow start phase again.

The value of *RTO* depends on another variable *RTT*, which is the round-trip time of the connection, i.e., the time it takes for the segment to travel from the sender to the receiver plus the time it takes for the ACK to travel from the receiver to the sender. Because real *RTT* may vary every time, Reno estimates the mean (smoothed) *RTT* via a low-pass filter. If we assume that *S_RTT* is this smoothed *RTT* estimate and *M_RTT* is the most recently measured *RTT*, *RTO* can be calculated as Listing 2.4.

Listing 2.4: RTO estimation

$$\begin{aligned}
&RTO = S_RTT + 4 \times v; \\
\text{where,} \\
&S_RTT = \alpha \times M_RTT + (1 - \alpha) \times S_RTT; \\
&v = \beta \times |M_RTT - S_RTT| + (1 - \beta) \times v;
\end{aligned}$$

Here v is the smoothed mean deviation of RTT . α is usually set to $1/8$, and β is set to $1/4$.

2.1.1.5 TCP Reno Variants

In Reno, if one packet is lost in a single window of data, after the lost packet is retransmitted to the receiver, the sender will receive a new ACK which acknowledges all of the packets transmitted before Fast Retransmit was entered, as shown in Figure 2.3. However, in the case of multiple packets dropped from a single window of data, this new ACK will only acknowledge the packets that are received before the second loss. The left packets transmitted before Fast Retransmit is not acknowledged by this ACK. We call this kind of ACK a partial ACK. Partial ACKs bring the sender out of fast recovery in Reno, resulting in a timeout. This will cause poor throughput in Reno. This throughput degradation has been observed in [47][48]. In order to quickly recover multiple packet losses, there are a number of enhanced loss recovery algorithms proposed such as NewReno [49], SACK [48][50], and FACK [51].

- **TCP NewReno**

In NewReno, when a sender receives a partial ACK, it does not come out of fast recovery. Instead, it assumes that the segment immediately after the most recently acknowledged segment has been lost, and hence the lost segment is retransmitted. Thus, in a multiple segment loss scenario, NewReno does not wait for a retransmission timeout and continues to retransmit lost segments every time it receives a partial ACK. This brings more efficiency in packet loss recovery.

- **TCP SACK**

Another way to deal with multiple segment losses is to tell the sender which segments have arrived at the receiver. Selective Acknowledgments (SACK) does exactly this. The receiver uses ACK with SACK option to indicate to the sender one contiguous block of data that has been received out of order at the receiver.

When SACK blocks are received by the sender, they are used to maintain an image of the receiver queue, i.e., which segments are missing and which have made it to the receiver. Scoreboard is set up to track transmitted and received packets according to the previous information of the SACK option. For each transmitted segment, scoreboard records its sequence number and a flag bit that indicates whether the segment has been “SACKed”. A segment with the SACKed bit turned on does not require to retransmit, but segments with the SACKed bit off and sequence number less than the highest SACKed segment are available for retransmission. Whether a SACKed segment is on or off, it is removed from the retransmission buffer only when it has been cumulatively acknowledged.

- **TCP FACK**

Forward Acknowledgments (FACK) is built on top of SACK. The name “Forward Acknowledgements” comes from the fact that the algorithm keeps track of the correctly received data with the highest sequence number. FACK provide a better way to estimate outstanding data more precisely, thus it is less bursty than SACK and can recover from episodes of heavy loss better than SACK. It also modifies the slow start phase of Reno, which decreases the congestion window more when a loss occurs during or near the slow-start period. Furthermore, it proposes a better way of halving the congestion window when congestion is detected: instead of immediately halving, the congestion window is decremented gradually.

2.1.1.6 Throughput Models for TCP Reno and its variants

Over last years, there have been many researchers working on providing analytical models for TCP Reno and its variants. Here we gives a brief review on these previous efforts.

In 1997, M. Mathis *et al.* [52] studied the long-term behavior of a TCP connection, and provided a simple throughput equation for TCP:

$$B(p) = \frac{S}{RTT} \frac{k}{\sqrt{p}} \quad (2.1)$$

where $B(p)$ is the throughput, S is the segment size, RTT is the round-trip time, K is a constant, and p is the packet loss rate. Note that this model only considers the AIMD algorithms during TCP communications.

In 1998, J. Padhye et al. derived a more complete model that includes the two types of loss indications used by TCP, namely triple duplicate ACK loss and timeout loss. Furthermore, it also considers the receiver window limitation during transmission. The derived equation is the Reno equation we mentioned in Chapter 1:

$$B(p) \approx \min \left(\frac{W_{\max}}{RTT}, \frac{1}{RTT \sqrt{\frac{2bp}{3}} + T_0 \min \left(1, 3 \sqrt{\frac{3bp}{8}} \right) p(1+32p^2)} \right) \quad (2.2)$$

where $B(p)$ is the throughput; $W_{\max}$ is the receiver window limitation; RTT is the round-trip time; b is a constant; p is the packet loss; T_0 is the RTO value. This equation is also used by S. Floyd in TFRC as a rate control protocol for streaming applications.

This model is further extended in [53] and [54] by including startup effects such as the connection establishment and the slow start, which have a significant impact on the latencies of short TCP transfers. Another model for short live TCP flows is also studied by [55].

All the above models are based on the drop-tail packet loss model [56]. The drop-tail packet loss model means: 1) the first packet in a round is lost with probability p ; 2) if the previous packet was not lost, the packet is lost with probability p ; 3) if a packet is lost,

then all subsequent packets in the round are lost. There is also another packet loss mode called the correlated packet loss. It means: 1) the first packet in a round is lost with probability p ; 2) if the previous packet was not lost, the packet is lost with probability p ; 3) if the previous packet was lost, the packet is lost with probability q . Papers [57] and [58] studied models based on this kind of packet loss.

There are also other efforts on modeling TCP performance. For example, paper [59] uses jump process driven Stochastic Differential Equations to model interactions of a set of TCP flows and Active Queue Management (AQM) routers such as Random Early Detection (RED) [60]; paper [61] provides a duality model for the interactions of TCP flows and network resources.

2.1.2 TCP Veno

Despite of the sophisticated recovery algorithms employed, TCP Reno and its variants do not modify the AIMD algorithms during transmission. That is to say, they always halve their window size whenever packet loss is detected by fast retransmit. This has caused negative effect of unnecessarily lowering the throughput when the loss is due to bit errors rather than network congestions. Particularly, in wireless networks with high bit error rates, this unnecessary throughput degradation is obvious.

TCP Veno deals with this wireless issue by modifying Reno's AIMD phase based on its state differentiator. Such state differentiator distinguishes congestive state and non-congestive state, and employs refined AIMD algorithms for different states.

2.1.2.1 Veno State Differentiator

Veno's state differentiator comes from the congestion detection method in TCP Vegas [62]. TCP Vegas is a quite different congestion control as compared to TCP Reno or its variants, because it can detect the early stage of congestion *before* packet loss occurs. However, Vegas has not been widely employed because it behaves conservatively in

CHAPTER 2. LITERATURE REVIEW

throughput when competing with the co-existing Reno-style connections [63][64]. Despite of its conservative behavior in congestion control, Vegas' congestion detection method does provide a clue on how to differentiate between congestive and non-congestive state.

The congestion detection method of Vegas works as follows. The sender measures the so-called *Expected* and *Actual* rates:

$$Expected = \frac{cwnd}{BaseRTT} \quad (2.3)$$

$$Actual = \frac{cwnd}{RTT} \quad (2.4)$$

where *cwnd* is the current congestion window size; *BaseRTT* is the minimum of the measured round-trip time collected so far, and is reset whenever packet loss is detected either due to time-out or duplicated ACKs; *RTT* is the smoothed round-trip time measured. The *Expected* rate can be seen as the best possible throughput the link can offer, and the *Actual* rate is the measured instant throughput during transmission. The difference of the rate is:

$$Diff = Expected - Actual \quad (2.5)$$

Vegas then uses the variable $N = Diff \times BaseRTT$ as a congestion detector. Specifically, if $N < \alpha$, Vegas considers the link is far away from congestion; if $N > \beta$ ($\beta > \alpha$), the link is going to be congestive; if $\alpha \leq N \leq \beta$, then the link is in an equilibrium point.

The implicit idea of Vegas is to maintain the queuing data in the bottleneck router buffer (N) to be between the lower threshold (α) and upper threshold (β). This can be seen as follows. When the queue size is zero, the *RTT* is *BaseRTT*. *BaseRTT* is basically the sum of the transmission delays and propagation delays throughout the forward and reverse paths (with queuing delay equal to zero). Then, when the packets of the connection accumulate at the queue, $RTT > BaseRTT$, and we have:

$$RTT = BaseRTT + \frac{N}{Actual} \quad (2.6)$$

CHAPTER 2. LITERATURE REVIEW

Here N is the queuing data size. That is, the extra delay of RTT is due to the queuing delay. Rearranging, we have:

$$\begin{aligned} N &= Actual \times (RTT - BaseRTT) \\ &= Diff \times BaseRTT \end{aligned} \tag{2.7}$$

which is just the variable used in Vegas.

Fundamentally different from TCP Vegas, which use N to constraint window fluctuation around the system equilibrium, TCP Veno only uses it to estimate network state – congestive or non-congestive. Specifically, if $N < \beta$ we can declare that the connection is not experiencing congestion and the network is in non-congestive state. Packet losses incurred in this state are most likely due to non-congestion reasons, e.g., bit errors. If $N \geq \beta$, on the other hand, the network is in congestive state and packet losses are considered to be induced by congestion reason.

By using this implicit state indication, TCP Veno adopts a more flexible AIMD mechanism than Reno does, which can improve TCP throughput over wireless networks.

2.1.2.2 Modified Additive-increase (AI) Algorithm in TCP Veno

Veno refines the additive-increase algorithm of Reno by forcing the TCP connection to stay longer at the operating region in which the bandwidth is fully utilized. Specifically, if $N < \beta$ where the network is detected as in non-congestive state and the available bandwidth is considered under-utilized, Veno increases its congestion window by $1/cwnd$ for every new ACK received; if $N \geq \beta$ where the network is congestive and the available bandwidth is considered fully utilized, then Veno increase its congestion window by $1/cwnd$ for every *other* new ACK received. The pseudo-code is shown in Listing 2.5.

Listing 2.5: Modified AI phase in TCP Veno

```

if ( $N < \beta$ )
     $cwnd = cwnd + 1/cwnd$  for every new ACK received;
else if ( $N \geq \beta$ )
     $cwnd = cwnd + 1/cwnd$  for every other new ACK received;

```

CHAPTER 2. LITERATURE REVIEW

2.1.2.3 Modified Multiplicative-decrease (MD) Algorithm in TCP Veno

The main improvement of Veno in throughput over wireless networks attributes to the modified multiplicative-decrease algorithm. Veno makes the congestion window cut down less (i.e., $1/5$) when packet loss occurs in the non-congestive state, thus it improves Reno's throughput during transmission. Specifically, if $N < \beta$ where the network is in non-congestive state, packet losses incurred in this state are most likely due to non-congestion reasons, so Veno sets *ssthresh* to be $4/5$ of *cwnd* and enters MD and FF phases; if $N \geq \beta$ where the network is in congestive state, packet losses are considered to be induced by congestion reasons, so Veno sets *ssthresh* to be $1/2$ of *cwnd* and enters MD and FF phases. The pseudo-code is shown in Listing 2.6. Note that here the code includes MD and FF phases.

Listing 2.6: Modified MD phase in TCP Veno

```

1) Retransmit the missing packet, and
   if ( $N < \beta$ )
       ssthresh = cwnd  $\times$   $4/5$ ;
   else if ( $N \geq \beta$ )
       ssthresh = cwnd  $\times$   $1/2$ ;
   cwnd = ssthresh + 3;
2) Each time another dup ACK arrives,
   cwnd = cwnd + 1;
3) When the next ACK acknowledging new data arrives,
   cwnd = ssthresh;

```

2.1.2.4 TCP Veno Variants

Since Veno only modifies the AIMD phase of Reno, it can also be integrated with different recovery schemes such as those in NewReno, SACK or FACK. These variants, namely NewVeno, SACK Veno, and FACK Veno, are all elaborated in [65]. Experiments have shown that these variants can further improve Veno's throughput.

2.1.3 Other TCP Variants over Wireless Networks

- Snoop TCP

CHAPTER 2. LITERATURE REVIEW

Snoop TCP requires to install a module called snoop agent at the base station (BS), which resides on the border between wired and wireless networks. The snoop agent monitors TCP traffic passing through it and caches the new packets. Once a packet is informed by the receiver to be lost (e.g., three duplicated ACKs) and the agent is aware that the packet is lost between itself and the wireless host, it will quickly retransmit the lost packet cached before and suppress the duplicated ACKs from the sender. This prevents unnecessary congestion control invocations at the sender and improves the whole throughput of TCP.

- **TCP with ECN**

Explicit Congestion Notification (ECN) [66] is an extension proposed to RED. RED is an active queue management mechanism in routers, which detects congestion before the queue overflows and provides an indication of this congestion to the end nodes. A RED router signals incipient congestion to TCP by dropping packets probabilistically before the queue runs out of buffer space. RED router operates by maintaining two levels of thresholds minimum (*minth*) and maximum (*maxth*). It drops packets probabilistically if and only if the average queue size lies between the *minth* and *maxth*. ECN relies on extension to RED, which marks a packet instead of dropping it when the average queue size is between *minth* and *maxth*. Upon receipt of congestion marked packet, the TCP receiver informs the sender (in the subsequent acknowledgement) about incipient congestion, which in turn will trigger the congestion avoidance algorithm at the sender. With ECN notification, TCP only concerns the congestion losses, eliminating impacts from random losses. Thus its throughput can be improved over wireless networks.

- **TCP Westwood**

TCP Westwood is a sender-side modification of the TCP congestion window algorithm that improves upon the performance of TCP Reno in wired as well as wireless networks. The general idea of Westwood is to use the bandwidth estimate BWE to set the congestion window ($cwnd$) and slow start threshold ($ssthresh$) after a congestion episode. Specifically, BWE is estimated by the sender along a TCP connection by measuring and averaging the rate of returning ACKs. Whenever the sender perceives a packet loss (i.e. a timeout occurs or three duplicate ACKs are received), the sender uses the BWE to properly set the $cwnd$ and the $ssthresh$, rather than blindly halving the congestion window as Reno does. In this way TCP Westwood avoids overly conservative reduction of $cwnd$ and $ssthresh$ and thus improves its performance.

TCP Westwood has some variants such as TCP Westwood+ [67] and TCP Westwood-A [68]. TCP Westwood+ improves Westwood bandwidth estimation algorithm to avoid inaccurate estimation of BWE in the presence of reverse traffic due to ACK compression. TCP Westwood-A introduces TCP Westwood with agile probing, which helps Westwood deal well with highly dynamic bandwidth, large propagation time/bandwidth, and random loss in the current and future heterogeneous Internet.

• WTCP

Wireless Transmission Control Protocol (WTCP) is quite different to standard TCP algorithms. It uses rate control instead of window-based congestion control. Specifically, the sender transmits data at an advertised rate that is checked and controlled by the receiver. The receiver checks the traffic rate by measuring the ratio of the inter-packet arrival time at the receiver to inter-packet sending time at the sender. When congestion becomes imminent, this period will become longer,

and the traffic rate can be altered accordingly through the receiver's acknowledgements. WTCP can also distinguish between congestion losses and random losses by checking the inter-packet delay of packets arriving close to the lost ones. Thus it can achieve better performance over wireless networks.

- **TCP-AP**

TCP-AP (TCP with Adaptive Pacing) was proposed to improve the throughput of TCP over ad hoc networks. It paces the packets allowed to be sent out by the congestion window adaptively. To configure the pacing, a metric of the 4-hop propagation delay is evaluated. It is the time between the transmission of a packet by the TCP source node and its reception by the node four hops downstream. The 4-hop propagation delay is chosen because a transmission currently in progress is assumed to interfere within a range of four hops, given the theoretical and experimental analysis. TCP-AP can be regarded as a hybrid between a window-based and a rate-based approach, adding rate-based mechanisms to TCP in order to avoid large bursts of packets. As a result, it reduces the packet losses due to the interfere within adjacent nodes, and improve the throughput.

- **C^3 TCP**

C^3 TCP (Cross-layer congestion control) is also an enhanced congestion control for ad hoc networks. In C^3 TCP, the information of bandwidth and delay within an end-to-end link is obtained by cumulating the intermediate hops' measurements. So C^3 TCP needs a feedback field added to the link layer header. When an ACK is generated at the destination node, the measurements are gathered hop by hop from the intermediate nodes and stored in the feedback field. There is an additional module located under TCP in the sender's protocol stack, which can modify the advertised window field in the ACK to dynamically limit the sender's sending rate

based on the measurements collected. By gathering sufficient information about the network, C^3 TCP avoids sending burst traffic to the ad hoc network and thus improve its throughput.

- **TCP-HO**

TCP-HO (TCP HandOff) was proposed to deal with heterogenous mobile networks, where different kinds of handoffs may often occur. TCP-HO assumes that a mobile host is able to detect the completion of handoff immediately and has a coarse estimation of new wireless links bandwidth. When a mobile host detects handoff completion, it will immediately notify the server through two duplicate ACKs, whose TCP option also carries the bandwidth of new wireless link. After receiving this notification, the server begins to transmit immediately and keeps updating ssthresh according to the bandwidth from mobile host and its new RTT samples. This updating will be stopped after four RTT samples or after congestion is detected. TCP-HO avoids unnecessary timeout in TCP when handoff occurs and shifts TCP's live connection to the new network quickly. Experimental results show that TCP-HO improves TCP performance without adversely affecting cross traffic in a heterogeneous mobile environment.

2.2 End-to-end Congestion Controls for Streaming

Streaming applications are more sensitive to delays than to packet loss. This makes TCP unsuitable for them, because the reliability and ordering semantics in TCP increase end-to-end delays and delay variations during transmission. Moreover, the AIMD algorithms in TCP cause large and abrupt reductions in transmission rate on packet losses, which also brings problems in playing real-time video/audio. Unlike TCP that has dominated end-to-end congestion control for bulk data transfer, congestion control for streaming is

still an emerging and ongoing research. There have been many studies on equation based and other congestion control mechanisms [69][70][71][72][73]. This thesis studies one of the most popular and recent protocols – TCP-Friendly Rate Control (TFRC), which is based on the Reno equation and has been standardize in RFC 3448. This section also reviews some enhancements of TFRC over wireless networks, e.g., MULTFRC.

2.2.1 TFRC

In general, TFRC is an equation based rate control protocol. Figure 2.4 gives an overview of TFRC mechanisms.

Figure 2.4: Overview of TFRC mechanisms.

TFRC lets receiver estimate the packet loss rate p on the link (the estimation method will be detailed latter). The estimated loss rate p is returned to the sender through ACKs. After receiving the value of p , the sender adjusts its sending rate based on the Reno equation that can model TCP Reno throughput:

$$T_{calc} = \frac{S}{RTT \sqrt{\frac{2bp}{3}} + 3RTO \sqrt{\frac{3bp}{8}} p(1 + 32p^2)} \quad (2.8)$$

where T_{calc} is the expected sending rate in bytes/sec; S is the packet size in bytes; RTT is the round-trip time; p is the steady-state rate of loss event; RTO is the retransmit timeout value. Note that, TFRC sender does not need to retransmit any missing packets during the transmission.

Following parts describe some other important algorithms in TFRC.

2.2.1.1 Slow Start

Slow start in TFRC occurs when TFRC is initialized or restarted after an idle time. In the slow start procedure, TFRC doubles its sending rate of each RTT , until packet losses are informed by the receiver through ACKs. Once a loss occurs, the slow start is terminated and TFRC uses the Reno equation (2.8) to calculate its sending rate.

2.2.1.2 Estimations of RTT , RTO , and p

In order to use the Reno equation, several variables in the equation are required, such as RTT , RTO , and p .

- **Estimations of RTT and RTO**

The measurement of RTT in TFRC is similar as that in TCP, that is, TFRC uses a low-pass filter to estimate the smoothed round-trip time. However, the estimation of RTO in TFRC is quite simple because RTO only critically affects the sending rate when the packet loss rate is very high. TFRC calculates RTO as $RTO = 4 \times RTT$.

- **Estimation of p**

The packet loss rate p is another critical variable in the Reno equation. It is estimated at the receiver side as follows.

The receiver judges a packet is lost when three packets with higher sequence number than itself arrive. The term loss event refers to several packets lost within one round-trip time. Note that the subsequent losses following the first loss in the round-trip time are ignored, i.e., at most one loss event in one round-trip time. The term loss interval is defined as the number of packets between loss events. The value of a loss interval is obtained by subtracting the sequence number of the first

CHAPTER 2. LITERATURE REVIEW

lost packet in a loss event from the sequence number of the first lost packet in the subsequent loss event. Figure 2.5 illustrates the relationship between loss events and loss intervals.

Figure 2.5: Estimation of loss events.

Let A_i be the i th recent loss interval, and let w_i be the weight of A_i . The receiver calculates the average loss interval of the recent n loss intervals as follows:

$$A(1, n) = \frac{\sum_{i=1}^n w_i A_i}{\sum_{i=1}^n w_i} \quad (2.9)$$

In TFRC, the default value of n is 8, and $w_1 = w_2 = w_3 = w_4 = 1, w_5 = 0.8, w_6 = 0.6, w_7 = 0.4, w_8 = 0.2$. Let $w = 1 / \sum_{i=1}^n w_i = \frac{1}{6}$, we can rewrite equation (2.9) as

$$A(1, n) = w \sum_{i=1}^n w_i A_i \quad (2.10)$$

In calculating $A(1, n)$, equation (2.9) or (2.10) ignores the most recent loss interval A_0 , which is the number of received packets since the latest loss event. If there is no loss event in one round-trip time, A_0 will continue to increase. If A_0 is large enough, it means that there are no packet losses for a long time; therefore the average loss interval should be increased accordingly. Thus, TFRC also calculates $A(0, n - 1)$,

which covers A_0 , according to

$$A(0, n-1) = w \sum_{i=0}^{n-1} w_{i+1} A_i \quad (2.11)$$

The refined value of average loss interval, A , is determined by

$$A = \max(A(1, n), A(0, n-1)) \quad (2.12)$$

Finally, the loss event rate, p , is given by

$$p = \frac{1}{A} \quad (2.13)$$

2.2.1.3 No Feedback Timeout

TFRC maintains a timer called the nofeedback timer. This timer is reset whenever the sender receives feedback (ACK) from the receiver. If there is no feedback arrival until it expires, the timer triggers no feedback timeout, and TFRC cuts the sending rate in half. However, if the sender has been idle for a long time and the sending rate is reduced to two packets per RTT , the halving operation is stopped. This ensures that the sending rate is never less than two packets per RTT as a result of an idle period. Once a new feedback resumes reporting no losses, the sender enters slow start procedure and doubles its sending rate each RTT , as described in Section 2.2.1.1.

2.2.2 MULTFRC

TFRC inherits TCP throughput degradation over wireless networks due to its embedded Reno equation. MULTFRC is a kind of sender-side modifications of TFRC, which only changes the sender protocol stack of TFRC and provides significant improvement in throughput over wireless networks.

MULTFRC borrows the idea from MULTCP [74], which was originally used to improve TCP performance in high bandwidth-delay product networks. The authors of

CHAPTER 2. LITERATURE REVIEW

MULTFRC find that using one TFRC connection in streaming applications results in under-utilization of the wireless bandwidth. They then propose the use of multiple simultaneous TFRC connections for a given wireless streaming application. The basic idea behind MULTFRC is to measure the round-trip time and adjust the number of connections accordingly. Specifically, MULTFRC increases the number of connections n by α/n or decrease it by β depending on the rtt measurements, that is,

$$n = \begin{cases} n - \beta, & \text{if } ave_rtt - rtt_min > \gamma rtt_min \\ n + \alpha/n, & \text{otherwise} \end{cases}$$

where α , β and γ are preset constant parameters, and the default values are 1, 1, and 0.2 respectively; ave_rtt is the average round trip time; for a given route, rtt_min is the minimum round trip time for that route, i.e. with no queuing delay. As a result, under ideal conditions MULTFRC keeps increasing the number of connections to make ave_rtt as close as possible to $(1 + \gamma)rtt_min$ without exceeding it. Ideally, ave_rtt becomes larger than rtt_min if and only if the link is fully utilized, and the queue on bottleneck link router is built up, introducing additional queuing delay. Thus by evaluating the difference between ave_rtt and rtt_min , MULTFRC can detect whether link is fully utilized or not and adjust the number of connections accordingly. The authors also provide an enhancement of MULTFRC, called AIO-TFRC [75] to overcome the complexity issue of the MULTFRC implementation and undesirable results of quantization effect in MULTFRC. However, in spite of high throughput achieved, both MULTFRC and its variant suffer from poor fairness and poor TCP-friendliness, as what MULTCP has encountered.

2.2.3 Other TFRC Variants over Wireless Networks

- **ECN-based TFRC**

ECN-based TFRC uses intermediate routers to detect incipient network congestion and inform the receiver using Explicit Congestion Notification (ECN) marking.

The receiver then measures the marking event rate p and feeds this information back to the sender. In this way, ECN-based TFRC ignores the losses occurring over the wireless hop or the effect of packet reordering encountered in the Internet, and only accounts congestion losses for sending rate adjustment.

- **Proxy-based TFRC**

Proxy-based TFRC employs a proxy to split the client-server connection, and uses TFRC only over the shared part of the client-server connection, i.e., the part between server and proxy. In the part between client and proxy, it tunnels RTP via a TCP which is TCP-friendly by definition. This method improves the average throughput and wireless link utilization.

- **WM-TFRC**

WM-TFRC uses the access point (AP) in wireless LAN to measure the rate of wireless loss events (i.e., loss events caused by bit error) p_w and feeds back its value to the sender periodically. Meanwhile, the receiver also provides feedback about the rate of total loss events (including wireless loss and congestion loss) p to the sender. Therefore, the sender can deduce the rate of congestion loss events (i.e., loss events caused by congestion) p_{con} by subtracting p_w from p . In this case, the sending rate of WM-TFRC, which is deduced from p_{con} instead of p , can be improved.

- **AED-based TFRC**

AED-based TFRC uses a new differentiation scheme called Accurate and Explicit Differentiation (AED) to improve the performance of TFRC. AED assumes that agents are deployed before and after each wireless link. The agents snoop through each packet and detect a loss by finding a packet with an out-of-order sequence number. Agents at the end of wired networks treat a packet loss as congestion

CHAPTER 2. LITERATURE REVIEW

loss and agents after a wireless link treat a packet loss as wireless loss. The agents then mark the packets that are not lost with information about the lost packets (i.e., whether those packets were lost due to congestion or bit error). Thus, when receiving these marked packets, the receiver can calculate the correct loss event rate and provide feedback to the sender.

Chapter 3

Analysis and Enhancements of End-to-end Congestion Controls for Bulk Data Transfer over Wireless Networks

This chapter presents our work on end-to-end congestion controls for bulk data transfer. We first derive a throughput model for TCP Veno, giving an in-depth analysis on TCP Veno behavior. Then we provide two enhancements for TCP Veno to further enhance its performance over two different types of wireless networks. Simulations and real network experiments results are also demonstrated.

3.1 Throughput Model for TCP Veno

In this work, we develop a throughput model for TCP Veno congestion control. The derived throughput equation, called Veno equation, can model TCP Veno throughput for bulk data transfer over different network scenarios, ranging from very low lossy links to very heavy lossy links. The simulation and live Internet results verify the effectiveness of such TCP Veno equation in the throughput modeling under different lossy situations.

3.1.1 VenO Equation

As described in Section 2.1.2, TCP VenO only refines the AIMD mechanism of Reno. The other parts of Reno, including the slow start, fast retransmit, fast recovery, and computation of the retransmission timeout, remain intact.

Here we simplify the AIMD mechanism of TCP VenO as follows:

in the AI phase,

$$cwnd = \begin{cases} cwnd + \frac{1}{cwnd}, & \text{for every new ACK} & N < \beta \\ cwnd + \frac{1}{cwnd}, & \text{for every other new ACK} & N \geq \beta \end{cases}$$

and in the MD phase,

$$cwnd = \begin{cases} cwnd \times \theta_1 & N < \beta \\ cwnd \times \theta_2 & N \geq \beta \end{cases}$$

Note that here we use θ_1 and θ_2 to replace the original fixed value in TCP VenO (i.e., $4/5$ and $1/2$), in order to derive a more general model which can characterize the effect of different congestion window drop factors.

Figure 3.1: Congestion window evolutions of VenO and Reno.

As shown in Figure 3.24, although the AI phase in VenO brings certain improvement in throughput of Reno, the most enhancement is attributed to VenO's smart MD phase with dynamic window reductions. To simplify the problem, we ignore the slightly different AI phase between VenO and Reno in the following derivation. Thereafter, VenO acts quite

similarly as Reno, and the only difference between Veno and Reno is how much the *cwnd* value should be cut down when losses occur: Reno always decreases its *cwnd* with factor $\frac{1}{2}$ while Veno sometimes decreases it with factor θ_1 (random loss), and sometimes with factor θ_2 (congestion loss). In this case, we introduce a random variable λ to represent a dynamic window drop factor, rather than a fixed factor of $\frac{1}{2}$ seen in [11]. That is, λ can be θ_1 or θ_2 with certain probability. In fact, λ reflects how Veno takes window multiplicative reduction in response to packet loss.

With reference to [11], we also divide the derivation of the Veno equation into three steps: firstly we only consider the situation where packet losses are indicated by the triple duplicated ACKs. We call these losses triple-duplicated (TD) losses; then we include packet losses indicated by time outs, which are called time-out (TO) losses; finally we extend the previous model to include the impact of the advertisement window limitation.

3.1.1.1 Veno Equation for TD Losses Only

Following [11], we consider a TD period (TDP) in Veno, which is a period between two TD losses. As shown in Figure 3.2, the y-axis represents the number of packets sent during the current TDP, and the x-axis represents the number of rounds in the current TDP. Here a “round” starts with the back-to-back transmission of W packets, where W is the current size of the TCP congestion window. Once all packets falling within the congestion window have been sent in this back-to-back manner, no other packets are sent until the first ACK is received for one of these W packets. This ACK reception marks the end of the current round and the beginning of the next round. In Figure 3.2 the number of rounds in a TDP is represented by X_i . b is the number of packets that are acknowledged by a received ACK. Many TCP receiver implementations send one cumulative ACK for two consecutive packets received (i.e., delayed ACK), so b is typically 2. If W packets are sent in the current round and are all received and acknowledged correctly, then W/b

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

ACKs will be received. Since each ACK increases the congestion window size by $1/W$, in the current round the window size is incremented by $W/b \times 1/W = 1/b$. W_{i-1} and W_i are the congestion window sizes at the ends of the previous TDP and the current TDP respectively. According to the Veno algorithm, the current TDP starts with the congestion window size of $\lambda_i W_{i-1}$, where λ_i is either θ_1 or θ_2 . After that, the window size is incremented by $1/b$ at each round until the end of the TDP.

Figure 3.2: Packets sent during a TD period.

If Y_i is the number of packets sent in the period, and A_i is the duration of the period, then the long-term steady-state Veno throughput can be shown to be:

$$B = \frac{E[Y]}{E[A]} \quad (3.1)$$

It is observed in [11] that:

$$E[Y] = \frac{1-p}{p} + E[W] \quad (3.2)$$

and,

$$E[A] = RTT(E[X] + 1) \quad (3.3)$$

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

where p is the probability that a packet is lost, given that either it is the first packet in its round or the preceding packet in its round is not lost. Note that p includes both congestion and random losses. RTT is the average value of round trip time. Detailed derivation can be found in [11]. According to Figure 3.2 we have:

$$W_i = \lambda_i W_{i-1} + \frac{X_i}{b} \quad (3.4)$$

Then the packets transmitted between these two TD losses are:

$$\begin{aligned} Y_i &= \sum_{k=0}^{X_i/b-1} (\lambda_i W_{i-1} + k)b + \eta_i \\ &= \lambda_i W_{i-1} X_i + \frac{X_i}{2} \left(\frac{X_i}{b} - 1 \right) + \eta_i \\ &= \frac{X_i}{2} (\lambda_i W_{i-1} + W_{i-1} - 1) + \eta_i \end{aligned} \quad (3.5)$$

where η_i is the number of packets sent in the last round, as seen in Figure 3.2. η_i is considered as a uniformly distributed random variable between 1 and W_i in [11], here we inherit this distribution in the Venio derivation.

From (3.4), (3.5) and (3.2), we have:

$$(1 - E[\lambda])E[W] = \frac{E[X]}{b} \quad (3.6)$$

and,

$$\frac{1-p}{p} + E[W] = \frac{E[X]}{2} ((1 + E[\lambda])E[W] - 1) + E[\eta] \quad (3.7)$$

where $E[\eta] = \frac{E[W]}{2}$.

Let $\gamma = E[\lambda]$, and from (3.6) and (3.7) we can get:

$$E[W] = \frac{b(1-\gamma)+1}{2b(1-\gamma^2)} + \sqrt{\frac{2(1-p)}{b(1-\gamma^2)p} + \frac{\left(\frac{b(1-\gamma)+1}{2}\right)^2}{b^2(1-\gamma^2)^2}} \quad (3.8)$$

Then from (3.6) and (3.3), we have:

$$E[X] = \frac{b(1-\gamma)+1}{2(1+\gamma)} + \sqrt{\frac{2b(1-\gamma)(1-p)}{(1+\gamma)p} + \frac{\left(\frac{b(1-\gamma)+1}{2}\right)^2}{(1+\gamma)^2}} \quad (3.9)$$

and,

$$E[A] = RTT \left(\frac{b(1-\gamma)+1}{2(1+\gamma)} + \sqrt{\frac{2b(1-\gamma)(1-p)}{(1+\gamma)p} + \frac{\left(\frac{b(1-\gamma)+1}{2}\right)^2}{(1+\gamma)^2}} + 1 \right) \quad (3.10)$$

Finally, substituting $E[A]$ and $E[W]$ into (3.1) and (3.2) we have:

$$B(p) = \frac{\frac{1-p}{p} + \frac{b(1-\gamma)+1}{2b(1-\gamma^2)} + \sqrt{\frac{2(1-p)}{b(1-\gamma^2)p} + \frac{\left(\frac{b(1-\gamma)+1}{2}\right)^2}{b^2(1-\gamma^2)^2}}}{RTT \left(\frac{b(1-\gamma)+1}{2(1+\gamma)} + \sqrt{\frac{2b(1-\gamma)(1-p)}{(1+\gamma)p} + \frac{\left(\frac{b(1-\gamma)+1}{2}\right)^2}{(1+\gamma)^2}} + 1 \right)} \quad (3.11)$$

For small values of p , the above can be approximated by:

$$B(p) \approx \frac{1}{RTT} \sqrt{\frac{(1+\gamma)}{2b(1-\gamma)p}} \quad (3.12)$$

3.1.1.2 Venet Equation for Both TD and TO Losses

So far equation (3.12) gives the Venet equation when only TD losses occur. Actually there are many TO losses over real communications, and they must be included in the model. According to [11], the TCP throughput with TO losses can be written as follows:

$$B = \frac{E[Y] + Q \times E[R]}{E[A] + Q \times E[Z^{TO}]} \quad (3.13)$$

where $E[Y]$ and $E[A]$ have been derived in the previous section. Q is the probability that a loss indication ending a TD period is a TO, $E[R]$ is the mean number of packets sent during TO periods, and $E[Z^{TO}]$ denotes the mean duration of TO periods.

As we have pointed out before, Venet does not change Reno's algorithms on dealing with TO losses, such as the computation of the timeout, and the backoff algorithm. We can easily derive the equation following the same routine described in [11], because the expressions of Q , $E[R]$, and $E[Z^{TO}]$ are the same in both Venet and Reno. Therefore, we have:

$$B(p) = \frac{\frac{1-p}{p} + E[W] + \hat{Q}(E[W])\frac{1}{1-p}}{RTT(E[X] + 1) + \hat{Q}(E[W])T_0\frac{f(p)}{1-p}} \quad (3.14)$$

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

where $E[W]$ and $E[X]$ have been given in the previous section. Note that, both $E[W]$ and $E[X]$ now include a new parameter γ here. T_0 denotes the period the sender will wait for to retransmit non-acknowledged packet,

$$\hat{Q}(w) = \min(1, \frac{(1 - (1 - p)^3)(1 + (1 - p)^3(1 - (1 - p)^{w-3}))}{1 - (1 - p)^w})$$

and,

$$f(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6$$

For small values of p , (3.14) is reduced to:

$$B(p) \approx \frac{1}{RTT \sqrt{\frac{2b(1-\gamma)p}{1+\gamma}} + T_0 \min\left(1, 3\sqrt{\frac{b(1-\gamma^2)p}{2}}\right)p(1+32p^2)} \quad (3.15)$$

3.1.1.3 Venet Equation with Impact of Window Limitation

Finally, we consider the Venet equation with the impact of advertisement window limitation. In such scenario, the congestion window will start to increase after the previous loss, and will keep constant when it reaches the advertisement window limitation. Let $W_{\max}$ denote the advertisement window limitation, U_i denote the number of rounds where the congestion window increases, and V_i denote the number of rounds where the congestion window keeps constant. Due to the new MD phase in Venet, $W_{\max}$ is given by [11]:

$$W_{\max} = \lambda_i W_{\max} + \frac{U_i}{b} \quad (3.16)$$

where λ_i is the proportion of *cwnd* value after the previous TD loss (θ_1 or θ_2), as defined in Section 3.1.1.1. We assume $E[U] = \frac{bW_{\max}}{2}$, same as that in [11].

Then considering the number of packets sent in the i th TD period, we have:

$$Y_i = \frac{U_i}{2}(\lambda_i W_{\max} + W_{\max}) + V_i W_{\max} \quad (3.17)$$

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

and then,

$$\begin{aligned} E[Y] &= \left(\frac{1+\gamma}{2}\right)W_{\max}E[U] + W_{\max}E[V] \\ &= \frac{b(1+\gamma)}{4}W_{\max}^2 + W_{\max}E[V] \end{aligned} \quad (3.18)$$

After getting $E[Y]$, we can easily derive the final throughput equation when the advertisement window is limited, following the steps of [11]:

$$B(p) = \frac{\frac{1-p}{p} + W_{\max} + \hat{Q}(W_{\max})\frac{1}{1-p}}{RTT\left(\frac{b(1-\gamma)}{4}W_{\max} + \frac{1-p}{pW_{\max}} + 2\right) + \hat{Q}(W_{\max})T_0\frac{f(p)}{1-p}} \quad (3.19)$$

Combine equation (3.14) and (3.19) we can get the final approximated result for small values of p :

$$B(p) \approx \min\left(\frac{W_{\max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2b(1-\gamma)p}{1+\gamma}} + T_0 \min\left(1, 3\sqrt{\frac{b(1-\gamma^2)p}{2}}\right)p(1+32p^2)}\right) \quad (3.20)$$

3.1.1.4 Expression of γ

Now we need to determine the value of γ . From our definition:

$$\begin{aligned} \gamma &= E[\lambda] \\ &= \theta_1 \times P(N < \beta) + \theta_2 \times P(N \geq \beta) \\ &= \theta_1 \times P(N < \beta) + \theta_2 \times (1 - P(N < \beta)) \\ &= \theta_2 + (\theta_1 - \theta_2) \times P(N < \beta) \end{aligned} \quad (3.21)$$

According to [20]:

$$N = \frac{W}{RTT} \times (RTT - BaseRTT) \quad (3.22)$$

where W is the current congestion window size, $BaseRTT$ is the minimum of the measured round-trip time, and RTT is the smoothed measured round-trip time. For simplicity, we assume the value of congestion window at losses (including congestion and

random losses), W , is uniformly distributed between 0 and $W_{\max}$, where $W_{\max}$ is the maximum congestion window size during the transmission. Then we have:

$$\begin{aligned} P(N < \beta) &= P(W < \beta \times \frac{RTT}{RTT - BaseRTT}) \\ &= \min \left(1, \frac{\beta \times RTT}{(RTT - BaseRTT) \times W_{\max}} \right) \end{aligned} \quad (3.23)$$

From (3.21) and (3.23) we have:

$$\gamma = \min \left(\theta_1, \theta_2 + (\theta_1 - \theta_2) \times \frac{\beta \times RTT}{(RTT - BaseRTT) \times W_{\max}} \right) \quad (3.24)$$

Equation (3.20) with (3.24) is the derived VenO equation.

3.1.1.5 Discussion

The VenO equation has a variable $\gamma \in [\theta_2, \theta_1]$ instead of the fixed factor $\frac{1}{2}$ in Reno equation. Here we call γ the “VenO parameter”. To determine an accurate expression for γ is still an ongoing work. Here we assume W is uniformly distributed. The validation on NS-2 simulation [76][77] and the live Internet experiments show that our assumption can approximate the experimental results quite well. We also assume that the refined AI phase in VenO has negligible improvement in TCP throughput as compared to the refined MD phase, which has been explained at the beginning of Section 3.1.1.

Other assumptions, same as those in [11], have also been made during our derivation:

1) our model does not include the slow start and fast recovery algorithm of VenO. We believe the impacts of both phases are small, and can be ignored in the model; 2) we assume that packet losses within a round are correlated. This is caused by the drop-tail policy of intermediate routers: if one packet is dropped by the full buffer, it is likely the following packets in the round will be dropped too; 3) we assume losses in one round are independent of losses in other rounds, because packets in different rounds are separated by one RTT and more, and thus they are likely to encounter different buffer states; 4) we assume the round trip time is independent of the window size.

3.1.2 Validation of Veno Equation

The correctness of the Veno equation is verified under both NS-2 simulations and the live Internet experiments. These quantitative results show that our Veno equation can accurately model TCP Veno's throughput.

3.1.2.1 Simulation Results

Figure 3.3 gives the topology of the NS-2 simulation experiment. The left side of the network has wired links with bandwidth of 10 Mbps, delay of 1 ms and buffer size of 50 packets. The right side of the network has wireless links with bandwidth of 10 Mbps, delay of 1 ms and buffer size of 50 packets. The bottleneck link between these two networks is a wired link and its bandwidth (W_b) and delay ($Delay_b$) will be varied in the experiments. Its buffer size is set to be 20 packets. TCP packets are transferred from the wired network to the wireless network. Packet size is 1000 bytes. All the simulation will run for 300 s and will be repeated 20 times. We plot the average of these results in the following figures and use the error bars to represent 95% confidence intervals.

Figure 3.3: Topology of NS-2 simulation.

In order to fully verify whether the Veno equation is able to accurately predict TCP Veno throughput or not, our simulation uses two different random loss models – exponential distribution and two-state distribution – in the wireless links. In the exponential

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

3.4.a: $W_b = 2Mbps$, $Delay_b = 80ms$.

3.4.b: $W_b = 2Mbps$, different delays.

Figure 3.4: Simulation results under exponential random loss: single flow.

distribution, packet loss rate ranges from 0.0001 to 0.1. In the two-state distribution, the good state has packet loss rate of 0.00001 and period of 8 s, and the bad state has packet loss rate ranging from 0.0001 to 0.1 and period of 4 s. Furthermore, the good state has 95% probability to transit to the bad state after every period, so does the bad state.

At first one TCP Veno connection is established. Figure 3.4.a shows the validation results under exponential random losses. Here the bandwidth of the bottleneck link is set to 2 Mbps and the delay is 80 ms. We also plot the curves when $\gamma = \theta_1 = \frac{4}{5}$ and $\gamma = \theta_2 = \frac{1}{2}$, which are the maximum and minimum boundaries of our equation respectively. Note that in this and the following experiments, we collect the dynamic values of RTT and $(RTT - BaseRTT)$ at every TD or TO, to calculate Veno parameter γ_i using equation (3.24). Finally we use the average of γ_i to calculate the result. As shown in Figure 3.4.a, most experimental results fall between the maximum and minimum boundaries, and they can be accurately predicted by the Veno equation. Figure 3.4.b plots the results under different bottleneck link delays.

Figure 3.5 with the two-state random loss model demonstrate that the theoretical results are quite consistent with the simulation results.

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

3.5.a: $W_b = 2Mbps$, $Delay_b = 80ms$.

3.5.b: $W_b = 2Mbps$, different delays.

Figure 3.5: Simulation results under two-state random loss: single flow.

3.6.a: UDP background traffic.

3.6.b: TCP Sack background traffic.

Figure 3.6: Simulation results under exponential random loss: background traffic.

Next, we introduce multiple UDP or TCP SACK connections as background traffic into our test. When UDP flows are added, there is only one TCP VenO connection. The bandwidth of the bottleneck link is set to 2 Mbps and the delay is 80 ms. UDP flows follow pareto distributions [78]. The packet size of UDP is 512 bytes. The burst time and idle time are both 100 ms. The rate of each connection varies from 250 Kbps to 1 Mbps. When TCP Sack flows are added, they range from 2 to 16, and there are equal numbers of TCP VenO flows running simultaneously. The bandwidth of the bottleneck

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

Figure 3.7: Simulation results under two-state random loss: background traffic.

link is set to 12 Mbps and the delay is 80 ms.

Carefully observing Figure 3.6 (with exponential random losses model introduced), and Figure 3.7 (with two-state random losses model introduced), we conclude that the theoretical analysis indeed match well with the simulation results, regardless of the types of losses.

Figure 3.8: More complex topology of NS-2 simulation.

To compare the theoretical analysis and simulation results in a more realistic scenario, a more complicated simulation topology is used here to validation the Veno equation.

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

3.9.a: A single flow.

3.9.b: Multiple flows.

Figure 3.9: Simulation results on a complicated topology.

Figure 3.8 illustrates the topology, where eight TCP Veno flows (Flow 1 ~ 8) run from the left side network to the right side network through the bottleneck link. The left side network has wired links with bandwidth of 10 Mbps, delay of 1 ms. The right side network is a heterogeneous network with wired and wireless links, whose bandwidths are all 10 Mbps. The delays of the wired links are $(20 \times i - 19)$ ms, $i = 1, 2, \dots, 8$, and the delays of the wireless links are all 1 ms. Random losses in wireless links follow exponential distribution and the packet loss rate ranges from 0.0001 to 0.1. The bottleneck link between these two networks is a wired link with bandwidth of 12 Mbps and delay of 10 ms. Its packet buffer size is set to be 40. Packet sizes of TCP Veno flows are 1000 bytes. We also set up 8 UDP connections as background traffic in our test. These UDP flows follow pareto distributions. The burst time and idle time are both 100 ms. The rate of each connection is 300 Kbps and the packet size is 512 bytes.

Figure 3.9.a gives the validation result on one of the TCP Veno flows – Flow 5. The theoretical results are quite consistent with the experimental results. Figure 3.9.b plots the validation results of Flow 2, 4, 6, 8. Obviously, all these experimental results can be accurately predicted by the Veno equation.

3.1.2.2 The Live Internet Experiments

In the live Internet experiments, we set up a wireless TCP Veno client at Nanyang Technological University (NTU) in Singapore, and a TCP Veno server at The University of Hong Kong (HKU) in Hong Kong. Thus the client and the server is connected via cross-country WAN, as shown in Figure 3.10.

Figure 3.10: Topology of the Internet experiments.

One TCP Veno flow runs over the link from the server to the client, and the packet size is 512 bytes. In the wireless side, the client is a Compaq laptop with W200D wireless card. The wireless Access Point is Cisco Linksys WRT54G. They are placed in the different corners of a room and the distance between them is about 8 meters. The bandwidth of the wireless network is 11 Mbps and the frequency is 2.4 GHz. TCP Veno is implemented in FreeBSD 4.3. The measurement method in the Internet experiments is same as that described in [11]. The 1hr flow trace is divided into 36 consecutive 100-second intervals, and we count the number of packets transmitted, RTT , $BaseRTT$, and the packet loss rate p in every interval. Note that p is given as the total number of loss indications (TD and TO) divided by the number of packets transmitted. Then, we put the average values of RTT and $BaseRTT$ into the Veno equation to calculate the theoretical result.

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

3.11.a: Result 1.

3.11.b: Result 2.

3.11.c: Result 3.

3.11.d: Result 4.

Figure 3.11: The Internet experimental results.

Due to the space constraints, we only give four samples collected recently, as shown in Figure 3.11. The results demonstrate that the Veno equation can model TCP Veno's throughput fairly well.

3.1.3 Summary

In this work, we provide a model for TCP Veno algorithm, to characterize its dynamic reductions of congestion window over wireless networks. A simple throughput equation called Veno equation is derived. We introduce a Veno parameter γ into previous work on Reno [11]. Such parameter accurately characterizes the refined MD phase of Veno. Our

extensive experiments on both simulation and the Internet have validated the correctness of the equation. In the future work, further efforts are needed to refine the simple assumption about the uniformly distributed W in the deduction, and a more accurate expression of the Veno parameter γ will be studied.

3.2 TCP Veno Enhancement over Light-load Wireless Networks

This work observes that TCP Veno behaves conservatively over light-load wireless networks. A new variable, *congestion loss rate*, is introduced into Veno's algorithm. It helps Veno deal with random loss more intelligently, by keeping its congestion window increasing if the link load is in light state. The simulation and real network results demonstrate that, such enhancement can improve Veno's throughput up to 60% without any fairness or friendliness sacrificed.

3.2.1 TCP Veno With *Congestion Loss Rate*

Many experiments have proved Veno's better performance over Reno in wireless environments [20][29][79][80]. In this section, however, we present results showing that Veno still misses much available bandwidth, especially when the network load is light and random losses are pervasive. Further analysis points out that such conservation is due to Veno's "blind" reduction of congestion window when random loss occurs. A new variable, named *congestion loss rate*, is introduced in our proposal. It helps Veno act more appropriately in response to random losses and grab more available bandwidth on the link, without any fairness or TCP-friendliness sacrificed.

3.2.1.1 Conservation of TCP Veno over Light-load Wireless Networks

Figure 3.12 gives a NS-2 simulation topology. The left side network has wired links with bandwidth of 10 Mbps, delay of 1 ms and buffer of 50 packets. The right side network

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

has wireless links with bandwidth of 10 Mbps, delay of 1 ms and buffer of 50 packets. Random loss in wireless links follows exponential distribution. The bottleneck link's bandwidth and delay will change in our experiments. Its buffer is set to be 20 packets. TCP packets are transferred from the wired network to the wireless network. Packet size is 1000 bytes.

Figure 3.12: Topology in NS-2 experiments.

To simulate the light load of the bottleneck link, we let only one TCP Veno connection run over it. The delay of the bottleneck link is set to be 80 ms, and the bandwidth is 2 Mbps and 8 Mbps respectively.

Table 3.1 presents Veno's throughputs over 2 Mbps and 8 Mbps bottleneck link respectively, under different packet loss rates (Pr). It shows that when the packet loss rate is larger than 0.005, Veno running over 8 Mbps bottleneck link has almost the same throughput as that over 2 Mbps one. In other words, additional 6 Mbps bottleneck link is not used by Veno even when there are no other competing connections.

Table 3.1: Throughputs of Veno over 2Mbps and 8Mbps.

	Pr=0.001	Pr=0.005	Pr=0.01	Pr=0.05	Pr=0.1
Over 2Mbps (packets/s)	192.46	142.65	100.91	27.83	11.49
Over 8Mbps (packets/s)	320.12	153.19	104.95	28.45	11.86

The reason why Veno behaves so conservatively in bandwidth utilization can be easily found in its congestion window evolution, as shown in Figure 3.13 (the dotted line). In

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

light-load wireless networks, congestion losses are rare and random losses are dominating. These random losses make Veno frequently cut down its congestion window by $\frac{1}{5}$, and keep *cwnd* value at a low level. In other words, such frequent random losses prevent Veno touching the equilibrium point during transmission. As a result, Veno wastes much available bandwidth during transmission.

Figure 3.13: Window evolution of Veno-LL and Veno, packet loss rate = 0.01, bottleneck bandwidth = 8Mbps.

According to the above analysis, a “blind” reduction of congestion window for random loss regardless of its congestion context is harmful to the improvement of Veno’s throughput. A more “intelligent” algorithm is needed to deal with these random losses, rather than always cutting down $\frac{1}{5}$ congestion window.

3.2.1.2 Congestion Loss Rate

We keep Veno’s method to estimate the backlog (N) at the link and deduce a packet loss is congestion loss ($N \geq \beta$) or random loss ($N < \beta$). However, a new variable, called

Figure 3.14: Calculation of congestion loss rate.

congestion loss rate, is introduced to help VenO adjust its congestion window in smarter way. Note that *congestion loss rate* is calculated as follows:

Considering a sequence of random losses during transmission $\{T_i\}$, where T_i is the time at which random loss occurs, we count the number of congestion losses occurring between two consecutive random losses T_{i-1} and T_i , called C_i , as shown in Figure 3.14. Then *congestion loss rate* at this moment con_r_i can be calculated as:

$$con_r_i = \frac{C_i}{T_i - T_{i-1}} \quad (3.25)$$

If $con_r_i > con_r_{i-1}$, which means the rate of congestion loss occurrence increases since the last time, then we assume the network state is becoming congestive, and cut down $\frac{1}{5}$ of congestion window at random loss T_i . Otherwise if $con_r_i \leq con_r_{i-1}$, which means the congestion of networks is not becoming worse, then we keep the value of congestion window unchanged at T_i .

In summary, our enhancement of VenO for light-load networks (hereinafter, called “VenO-LL”) acts as follows:

$$cwnd = \begin{cases} cwnd, & N < \beta \text{ and } con_r_i \leq con_r_{i-1} \\ cwnd \times \frac{4}{5}, & N < \beta \text{ and } con_r_i > con_r_{i-1} \\ cwnd \times \frac{1}{2}, & N \geq \beta \end{cases}$$

As shown in Figure 3.25 (the solid line), over a light-load network con_r_i is always 0 during transmission, except of occasional pulses. Thus $cwnd$ value will keep increasing at most random losses and help VenO-LL absorb as much available bandwidth as possible.

It is worthy to emphasize that, as the network becomes more and more congestive, and congestion becomes the main reason of packet losses, our proposal turns to behave similarly as original VenO does. This can be observed in the performance evaluation part, where more detailed experiments are conducted.

3.2.2 Performance Evaluation

In this section, we evaluate VenO-LL's performance in throughput, fairness, and TCP-friendliness, as compared to VenO. The evaluation is based on both NS-2 simulations and real network experiments.

3.2.2.1 Simulation Results

The topology and settings are same as those in Figure 3.12. The bandwidth of bottleneck link is set to be 12 Mbps and the delay is 80 ms. Random loss rate in wireless links changes from 0.0001 to 0.1 in packet unit.

- **Throughput**

In each experiment, we at first set up N VenO connections running over the network for 300s, and calculate their average throughput. Here the throughput is counted as the number of packets sent divided by the transmission time. Then we set up the same number of VenO-LL connections running over the network for the same duration, and also calculate their average throughput. In order to better compare the difference of their throughput, we define the normalized throughput TH_n as following:

$$TH_n = \frac{TH_{Enhanced}}{TH_{VenO}} \quad (3.26)$$

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

where, $TH_{Enhanced}$ and TH_{Veno} are the average throughput of Veno-LL flows and Veno flows respectively. This metric illustrates the percentage of improvement Veno-LL can obtain as compared to Veno. The higher TH_n is, the greater the improvement is.

The number of connections N varies from 1 to 64 in the experiments, to make the bottleneck link more and more congestive.

Normalized Throughput

Figure 3.15: Throughput improvement of Veno-LL under different number of connections (N) and different random loss rates (Pr).

As shown in Figure 3.15, our proposal can effectively improve Veno's throughput when connection number is small (the improvement can be up to 60% when random loss is around 0.01). However, as the connection number increases, which means the network load increases, our proposal performs more and more similarly as Veno (TH_n is around 1) except under very heavy random losses. In other words, our proposal is a feasible enhancement of Veno over light-load wireless networks.

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

- **Fairness**

Fairness means the same kind of flows should share the total bandwidth fairly. To reflect the fairness more clearly, we use the Jain's Fairness Index f which is define in [81]:

$$f = \frac{\left(\sum_{i=1}^n x_i\right)^2}{n \sum_{i=1}^n x_i^2} \quad (3.27)$$

where, n is the number of connections, x_i is the throughput of the i th connection. The closer f is to 1, the more fairness that kind of flows has.

Figure 3.16: Fairness comparison between Veno-LL and Veno.

In each experiment, we at first set up N Veno connections running over the network for 300s, calculate the throughput of each connection, and get their Farness Index using equation 3.27. Then we set up the same number of Veno-LL connections running over the network for the same duration, and also calculate their Fairness Index. The number of connections N varies from 1 to 64 in the experiments.

Figure 3.16 plots the Fairness Index of Veno-LL and Veno respectively. The values f of Veno-LL and Veno in the figure are almost same in each scenario, which proves that our enhancement does not harm original Veno's fairness.

• TCP-friendliness

Veno has shown its compatibility to the dominating version of TCP today [20]. Here we study whether our proposal has the same TCP compatibility as Veno. We first set up certain numbers (ranging from 2 to 64) of TCP Sack flows over the link, and calculate their average throughput T_1 . Then we replace half of them with the objective flows (Veno-LL or Veno) and recalculate the average throughput of the left TCP Sack flows T_2 . We define the normalized throughput F as follows:

$$F = \frac{T_2}{T_1} \quad (3.28)$$

Figure 3.17: Friendliness comparison between Veno-LL and Veno.

If F is 1, it means Sack flows are not affected by the objective flows, and thus the objective flows are totally friendly. The closer F is to 1, the more friendliness the objective flows have.

As illustrated in Figure 3.17, the value F of Veno-LL is always around that of Veno in different scenarios, which means our proposal does not sacrifice Veno's TCP-friendliness when improving its throughput.

3.2.2.2 Real Network Experiments

We also conduct the experiment in a wireless local area network (WLAN). The topology of the WLAN is shown in Figure 3.18. The TCP senders use FreeBSD 4.3, in which Veno and Veno-LL are implemented. The TCP receivers are normal linux machines with kernel 2.6.12. They are connected by a Gateway with FreeBSD 4.9. Dummynet [82] is used in this Gateway to shape the bandwidth and set the random loss rate.

Figure 3.18: Topology of WLAN experiments.

The bandwidth of the bottleneck link is set to be 5 Mbps, and the delay is 60 ms. Furthermore, the random loss rate is set to be 0.005. Two cases are studied here based on different traffic load: light load case where there is no other background traffic, and heavy load case where there is an co-exist 3 Mbps UDP flow during transmission. In each case, different types of flows are tested for 300 s respectively: 1) four Veno connections (two in each sender PC); 2) two Veno and two Veno-LL connections (one Veno and one Veno-LL in each sender PC); 3) four Veno-LL connections (two in each sender PC). Figures 3.19 and 3.20 plot their sequence numbers against the time in each case.

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

Figure 3.19: Experiments over WLAN: light load case.

Figure 3.20: Experiments over WLAN: heavy load case.

From these figures, three important observations can be made. The first observation is about the fairness. Either in Figure 3.19.c or 3.20.c, the increasing rate in sequence number of each Enhance Veno flow is similar. This proves Veno-LL can share the network source fairly. The second observation is, Veno-LL has a similar increasing rate as Veno does when the network is in heavy-load state (Figure 3.20.b), but has a much higher increasing rate when the network is in light-load state (Figure 3.19.b). This proves Veno-LL has a better throughput than Veno over light-load wireless networks. The third observation is about the TCP-friendliness. Although Veno-LL has a higher increasing rate in the light-load networks, it does not affect the increasing rate of Veno running with them. This can be observed by comparing between Figure 3.19.a and 3.19.b. This

proves VenO-LL does not sacrifice any TCP-friendliness.

3.2.3 Summary

In this work, we introduce a new variable called *congestion loss rate* into TCP VenO. *Congestion loss rate* can indicate the trend of congestion occurrence on the link and avoid “blind” reduction of congestion window at random losses, especially when the network load is light. Simulation and real network results demonstrate such enhanced TCP VenO can obtain significant throughput improvement over TCP VenO without any fairness or TCP-friendliness sacrificed.

3.3 TCP VenO Enhancement over High Bandwidth-Delay Product Networks

This work focuses on another situation TCP VenO may encounter during transmission – high Bandwidth-Delay Product (BDP) network. We find that TCP VenO suffers severe throughput degradation under random losses in a high BDP network. To alleviate such suffering, a new variable named $pBDP$ is proposed to indicate the BDP state of the network. It helps VenO deal with random losses more intelligently, by keeping its congestion window increasing if the link is in the high BDP state. The experiments based on the simulation and local area network demonstrate that, such enhancement can improve VenO’s throughput up to 80% without any fairness or friendliness sacrificed.

3.3.1 TCP VenO with $pBDP$

The BDP is loosely defined as the product of the round-trip delay for a data connection and the capacity of the bottleneck link in its path. Note that here we use the term “high BDP” as that in [13], i.e., the BDP is larger than the buffer size on the bottleneck link. This definition covers very large scale in different types of networks, besides of the high

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

speed networks. In a high BDP network, our study shows that TCP VenO suffers severe throughput degradation under random losses. Our further analysis points out that, such throughput degradation in VenO is due to its “blind” reduction of congestion window when random losses occur, without concerning the specific state of the link at that time. Based on the observation in [13], a new variable named $pBDP$ is introduced into VenO. It indicates the BDP state of the link and helps VenO act more appropriately in response to random losses.

It is worthy to emphasize that, this work only focuses on alleviating the random loss impact on VenO over a high BDP network. Other applicable enhancements for high BDP networks, such as a faster additive-increase (AI) and a less multiplicative-decrease (MD) during the congestion window evolution [83][84], are out of the scope. However, our scheme can seamlessly cooperate with these enhancements.

3.3.1.1 Random Loss Impacts on TCP VenO over High BDP Networks

Figure 3.21 gives the topology of the NS-2 simulations used in this section. The left side network has wired links with bandwidth of 100 Mbps and delay of 1 ms. The right side network has wireless links with bandwidth of 100 Mbps and delay of 1 ms. Random loss in wireless links follows exponential distribution. The bottleneck link’s bandwidth and delay are varied in our experiments. Its buffer is set to be 50 packets. TCP packets are transferred from the wired network to the wireless network. Packet size is 1000 bytes. The duration time of each experiment is 300 s.

One TCP VenO connection is established in the experiment and its throughput is studied in three cases: at first we set the bottleneck bandwidth to be 2 Mbps, and its delay to be 50 ms; then we extend the delay to 300 ms; finally we set the bandwidth to be 12 Mbps and the delay to be 50 ms. The BDP of the bottleneck link is 25, 150, and 150 respectively. As compared to the buffer size (50), the latter two cases can be

Figure 3.21: Topology in NS-2 experiments.

regarded as the high BDP links. Note that, flow control is disabled in our experiment, thus the congestion window can increase as large as possible.

Figure 3.22 presents the results of Venos throughputs in these three cases under different random loss rates from 0.0001 to 0.1. As observed in the figure, when the random loss rate increases, Venos throughput decreases much faster in the two high BDP cases. For example, when the random loss rate increases from 0.0001 to 0.001, the decrease degree of Venos throughput is only 3% in the first case ($BDP = 25$), but is 52% and 47% in the latter two cases ($BDP = 150$) respectively.

The reason why Venos encounters such severe throughput degradation can be easily found in its congestion window evolution. Figure 3.23 plots Venos congestion window evolution over 2 Mbps, 50 ms link and 2 Mbps, 300 ms link respectively. According to [20], Venos cuts down $\frac{1}{5}$ of the window size in response to random losses. When the random loss rate is 0.0001 where the random losses are rare, the congestion window can always touch the equilibrium point and cause congestion losses. However, when the random loss rate is 0.001, many random losses prevent this happening and keep the congestion window size in a lower level than that under random loss rate of 0.0001. This makes Venos miss available bandwidth of the link. As shown in Figure 3.23.b, the gap between

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

Figure 3.22: Random loss impacts on Veno in three BDP cases.

the average window sizes under different loss rates becomes much larger when BDP is high. In other words, although Veno cuts down less window size ($\frac{1}{5}$) under random losses than Reno does, such action is still conservative in high BDP networks.

3.23.a: Over 2Mbps, 50ms bottleneck link.

3.23.b: Over 2Mbps, 300ms bottleneck link.

Figure 3.23: Congestion window evolutions of TCP Veno.

Therefore, we propose a new variable to indicate VenO about the BDP state of the network. It can help VenO deal with random losses more intelligently, rather than always cutting down $\frac{1}{5}$ of the congestion window.

3.3.1.2 $pBDP$

The random loss impact on TCP Reno over high BDP networks has been studied by [13] in both analytical and experimental way. It observes that, random losses lead to significant throughput deterioration when $p(\mu T)^2$ is large. Here p is the random loss probability, μ (packet/s) is the capacity of the bottleneck link, and $T = (\tau + \frac{1}{\mu})$ is the sum of propagation delay (τ) and service time ($\frac{1}{\mu}$). Typically $\tau \gg \frac{1}{\mu}$, so $T \approx \tau$. Following this work, we can use the variable $p(\mu\tau)^2$ to indicate the BDP state of the link. If it is larger than certain threshold, we consider the link a high BDP link and do not cut down the congestion window size when a random loss occurs. Otherwise, we cut down the congestion window by $\frac{1}{5}$ as the original VenO does. Note that here “BDP state” means the cooperative impacts from both BDP and random losses. For example, if the random losses is heavy, VenO’s throughput will go down severely even the BDP is low.

In a real network, however, the TCP sender is unable to exactly know the values of p , μ , and τ . An alternative method is proposed here to approximate the value of $p(\mu\tau)^2$.

The random loss rate p can be estimated by using a similar method as that in TFRC [9], as described in Section 2.2.1.2. VenO considers a packet loss is random loss when $N < \beta$. Then we define the term random loss event as that in TFRC. With reference to Figure 2.5, let A_i be the i th recent random loss interval, and let w_i be the weight of A_i , then the receiver calculates the average random loss interval of the recent n loss intervals as follows:

$$A = \frac{\sum_{i=1}^n w_i A_i}{\sum_{i=1}^n w_i} \quad (3.29)$$

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

The default value of n is 8, and $w_1 = w_2 = w_3 = w_4 = 1, w_5 = 0.8, w_6 = 0.6, w_7 = 0.4, w_8 = 0.2$. Finally, the estimated random loss event rate, $\bar{p}$, is given by

$$\bar{p} = \frac{1}{A} \quad (3.30)$$

The value $\mu\tau$ can be estimated as follows. Consider a network with a constant propagation delay τ and a buffer size B . If we ignore the processing time at the routers and the end hosts, the round-trip time (RTT) of a TCP flow is roughly the sum of the propagation delay and the queuing delay, i.e.,

$$RTT \approx \tau + \frac{b}{\mu} \quad (3.31)$$

where b is the number of packet cumulating in the bottleneck link buffer, and $0 \leq b \leq B$.

Thus we have

$$RTT_{min} \approx \tau \quad (3.32)$$

and,

$$RTT_{max} \approx \tau + \frac{B}{\mu} \quad (3.33)$$

On the other hand, according to [13], the maximal size of a TCP congestion window is

$$W_{max} = \mu\tau + B + 1 \approx \mu\tau + B = \mu\left(\tau + \frac{B}{\mu}\right) = \mu RTT_{max} \quad (3.34)$$

From (3.32) and (3.34), we have

$$\mu\tau \approx \frac{W_{max} RTT_{min}}{RTT_{max}} \quad (3.35)$$

where W_{max} , RTT_{min} , and RTT_{max} can be easily monitored in the TCP sender.

Figure 3.24 gives the validation of equation (3.35) under a simulation over 12 Mbps, 50 ms networks. Note that, the value of W_{max} will be reset to 0 after each time-out occurs. This makes our scheme adaptable to the dynamic of the network. As observed

Figure 3.24: Estimation of $\mu\tau$ under different random loss rates.

in the figure, the large random loss rate (0.01) will cause $\mu\tau$ to be underestimated. This is because the frequent random losses prevent the congestion window touching its maximal size, thus make the monitored W_{max} always lower than it should be. Although the underestimated value of $\mu\tau$ may make our scheme misinterpret some high BDP links to be normal links and obtain no improvement in these cases, it also avoids the trend of the scheme to be aggressive in most cases.

From (3.30) and (3.35), the variable indicating the BDP status of the network, named $pBDP$, is obtained by:

$$pBDP = \bar{p} \left(\frac{W_{max} RTT_{min}}{RTT_{max}} \right)^2 \quad (3.36)$$

In summary, our enhancement of Veno for high BDP networks (hereinafter, called “Veno-HB”) acts as follows:

$$cwnd = \begin{cases} cwnd, & N < \beta \text{ and } pBDP \geq \alpha \\ cwnd \times \frac{4}{5}, & N < \beta \text{ and } pBDP < \alpha \\ cwnd \times \frac{1}{2}, & N \geq \beta \end{cases}$$

According to [20] and [13], the default values of β and α are set to 3 and 10 respectively.

3.3.2 Performance Evaluation

We compare our Veno-HB with the original Veno mechanism based on the NS-2 simulations and the real network experiments.

3.3.2.1 Simulation Results

The experiment environments are same as those in Figure 3.21. Throughput, fairness, and friendliness of Veno-HB are evaluated here, as compared to those of Veno.

- **Throughput**

At first only a single connection is set up. Figure 3.25 presents the congestion window evolutions of Veno and Veno-HB over a bottleneck link of 2 Mbps, 300 ms. The random loss rate is 0.001. As shown in the figure, Veno-HB avoids many unnecessary window cuts under random losses, and keeps its congestion window in a high level near the BDP.

More complete study is shown in Figure 3.26. It plots the throughput enhancements of our proposal under different BDPs and different random loss rates from 0.0001 to 0.1. The normalized throughput (TH_n) is defined as equation (3.26). As shown in the figure, Veno-HB has a good improvement (up to 80%) in throughput, especially over high BDP networks.

Next, we increase the number of connections of Veno and Veno-HB from 1 to 64, and investigate the throughput enhancement under multiple connections scenarios. The bottleneck link here is set to be 12 Mbps and 50 ms. As shown in Figure 3.27, our proposal can effectively improve Veno's throughput when connection number is small. As the connection number increases, network resources are shared by multiple flows and now the network is no longer a high BDP network. Thus Veno-HB performs more and more similarly as Veno does (TH_n is around 1) except under very heavy random losses.

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

Figure 3.25: Congestion window evolutions of Veno-HB and Veno over 2Mbps, 300ms bottleneck link.

Normalized Throughput

Figure 3.26: Throughput improvement of Veno-HB under different BDP cases and different random loss rates (Pr).

Normalized Throughput

Figure 3.27: Throughput improvement of VenO-HB under multiple connections (N) and different random loss rates (Pr).

- **Fairness**

In the experiments, multiple VenO-HB and VenO connections (ranging from 1 to 64) are established respectively. The bottleneck link here is set to be 12 Mbps and 50 ms. To reflect the fairness of VenO-HB and VenO, we also use Jain's Fairness Index f which is define as equation (3.27). Figure 4.16 plots the Fairness Index of VenO-HB and VenO respectively. The values f of VenO-HB and VenO in the figure are almost same in each scenarios, which proves that our enhancement does not harm original VenO's fairness.

- **TCP-friendliness**

We first set up certain numbers (ranging from 2 to 64) of TCP Sack flows over the link, and calculate their average throughput T_1 . Then we replace half of them with the objective flows (VenO-HB or VenO) and recalculate the average throughput of

Figure 3.28: Fairness comparison between Veno-HB and Veno.

the remaining TCP Sack flows T_2 . We also define the normalized throughput F as equation (3.28), to reflect the TCP-friendliness of Veno-HB and Veno. Note that the bottleneck link here is set to be 12 Mbps and 50 ms.

As observed in Figure 4.17, the value F of Veno-HB is always around that of Veno in different scenarios, which means our proposal does not sacrifice Veno's friendliness when improving its throughput.

3.3.2.2 Real Network Experiments

We also implement this Enhanced Veno into FreeBSD 4.3, and run the experiment in a wireless LAN. The topology of the wireless LAN is same as that in Figure 3.18. The TCP senders use FreeBSD 4.3, in which Veno and Veno-HB are implemented. The TCP receivers are normal linux machines with kernel 2.6.12. They are connected by a Gateway with FreeBSD 4.9. Dummynet is used in this Gateway to shape the bandwidth and set the random loss rate.

Figure 3.29: Friendliness comparison between Veno-HB and Veno.

The random loss rate of the bottleneck is set to be 0.005 and the buffer size is set to be 50 packets. Three cases are studied here based on different pairs of bandwidth and delay: (2 Mbps, 40 ms), (2 Mbps, 240 ms), and (12 Mbps, 40 ms). In each case, different types of flows are tested for 300 s respectively: 1) four Veno connections (two in each sender PC); 2) two Veno and two Veno-HB connections (one Veno and one Veno-HB in each sender PC); 3) four Veno-HB connections (two in each sender PC).

From these figures, three important observations can be made. The first observation is about the fairness. Either in Figure 3.30.c, 3.31.c, or 3.32.c, the increasing rate in sequence number of each Enhance Veno flow is similar. This proves Veno-HB can share the network source fairly. The second observation is, Veno-HB has a similar increasing rate as Veno does when the network is not a high BDP network (Figure 3.30.b), but has a much higher increasing rate when the network is a high BDP network (Figure 3.31.b and 3.32.b). This proves Veno-HB has a better throughput than Veno over high BDP networks. The third observation is about the TCP-friendliness. Although Veno-HB has

CHAPTER 3. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR BULK DATA TRANSFER OVER WIRELESS NETWORKS

3.30.a: Four Venetian connections.

3.30.b: Two Venetian and two Venetian-HB connections.

3.30.c: Four Venetian-HB connections.

Figure 3.30: Sequence numbers against time over a 2Mbps, 40ms link.

3.31.a: Four Venetian connections.

3.31.b: Two Venetian and two Venetian-HB connections.

3.31.c: Four Venetian-HB connections.

Figure 3.31: Sequence numbers against time over a 2Mbps, 240ms link.

3.32.a: Four Venetian connections.

3.32.b: Two Venetian and two Venetian-HB connections.

3.32.c: Four Venetian-HB connections.

Figure 3.32: Sequence numbers against time over a 12Mbps, 40ms link.

a higher increasing rate in the high BDP networks, it does not affect the increasing rate of Veno running with them. This can be observed by comparing between Figure 3.31.a and 3.31.b, or between Figure 3.32.a and 3.32.b. This proves Veno-HB does not sacrifice any TCP-friendliness.

3.3.3 Summary

This work introduces a new variable called $pBDP$ into TCP Veno. It indicates the BDP state of the bottleneck link and avoids Veno’s “blind” reduction of congestion window at random losses. The simulations and local area network experiments demonstrate that, such enhanced TCP Veno can obtain significant throughput improvement over TCP Veno without any fairness or TCP-friendliness sacrificed. Furthermore, $pBDP$ can seamlessly cooperate with further enhancement of Veno over high BDP networks, such as a faster additive-increase (AI) and a less multiplicative-decrease (MD) during the congestion window evolution. A complete optimization of Veno over high BDP networks is under the future investigation.

Chapter 4

Analysis and Enhancements of End-to-end Congestion Controls for Streaming over Wireless Networks

In this chapter, we describe our work on the end-to-end congestion controls for streaming. We propose two enhancements of TFRC to improve its performance over wireless networks. One enhancement is equipped with the Veno equation rather than the Reno equation to adjust its sending rate, and another one uses Veno's state differentiator to alleviate the impact of random losses on the sending rate. Simulations and real network experiments results for these two proposals are also shown in this chapter.

4.1 Veno Equation Based TFRC Enhancement

As described in Section 1.2.2, TFRC inherits TCP throughput degradation over wireless networks due to its embedded Reno equation. In this thesis, a more advanced equation, called Veno equation, has been derived in Section 3.1.1. Note that the Veno equation is derived from the wireless TCP (TCP Veno) rather than the wired TCP (TCP Reno), thus it can model TCP behavior more appropriately over wireless networks. In this work, we choose to make use of the Veno equation to enhance TFRC over wireless networks. The enhancement is referred as TFRC Veno. Our extensive experiments including simulations

and the Internet measurements demonstrate that, TFRC VenO obtains two achievements: 1) it has a significant throughput improvement – in wireless networks with 10% loss rate, it can obtain 300% improvement over the original TFRC; 2) it inherits the desirable features of TFRC, namely, good fairness, nice TCP-friendliness and smoothness of sending rate. Moreover, TFRC VenO only needs to modify the sender-side protocol of TFRC while the receiver-side or intermediate node protocol stack remains intact.

4.1.1 TFRC VenO

Different from all the previous efforts that keep the Reno equation in TFRC intact, TFRC VenO tries to use the VenO equation to replace the Reno equation. The basic mechanism of TFRC has been described in Section 2.2.1. TFRC VenO modifies TFRC by simply replacing the Reno equation with the VenO equation, that is, the sender now adjusts its sending rate based on the following equation:

$$T_{calc} = \frac{S}{RTT \sqrt{\frac{2b(1-\gamma)p}{1+\gamma}} + 3RTO \sqrt{\frac{b(1-\gamma^2)p}{2}} p(1 + 32p^2)} \quad (4.1)$$

where T_{calc} is the expected sending rate in bytes/sec; S is the packet size in bytes; RTT is the round-trip time; p is the steady-state rate of loss event; RTO is the retransmit timeout value.

Note that here we ignore the effect of the maximum congestion window size in the VenO equation (3.20). Furthermore, the original VenO equation calculates throughput in packets/sec, while here we multiply it by the packet size (S) to calculate throughput in bytes/sec. These two changes are the same as those of the Reno equation when used in TFRC. The other mechanisms in TFRC, such as slow start, packet loss estimation, and time-out mechanism, are not changed in TFRC VenO. Here γ is in the range $[\theta_1, \theta_2]$, with the settings of $\theta_1 = 0.95$ and $\theta_2 = 0.5$.

4.1.2 Performance Evaluation

We conduct various experiments including NS-2 simulation and live Internet experiments, in order to comprehensively evaluate the performance of TFRC Veno in terms of throughput, fairness, TCP-friendliness, and sending rate variation. We also compare our proposal with the original TFRC and another sender-modified enhancement MULTFRC [35]. The results in the experiments show that, TFRC Veno combines both advantages of TFRC and MULTFRC while avoiding their respective drawbacks, that is to say, TFRC Veno has much better throughput than TFRC while keeping the nice features of fairness, TCP-friendliness, and sending rate variation; TFRC Veno has much better fairness and TCP-friendliness than MULTFRC while having the same throughput as MULTFRC can reach.

4.1.2.1 Simulation Results

The topology of the simulation experiments is depicted in Figure 4.1. The left side of the network has wired links with bandwidth of 10 Mbps, delay of 1 ms and buffer size of 50 packets. The right side of the network has wireless links with bandwidth of 10 Mbps, delay of 1 ms and buffer size of 50 packets. The bottleneck link between these two networks is a wired link with bandwidth of 12 Mbps, delay of 80 ms. Its buffer size is set to be 20 packets. Three types of flows – TFRC, MULTFRC, and TFRC Veno – are transferred from the wired network to the wireless network. Packet sizes are all 1000 bytes. Furthermore, we set up 8 UDP connections with pareto distribution over the bottleneck link as the background traffic. The burst time and idle time are both 100 ms. The packet size of UDP is 512 bytes and the rate of each connection is 300 Kbps. Each experiment lasts 300 s, and is repeated by 20 times. We plot the average of these results in the following figures and use the error bars to represent 95% confidence intervals.

In order to fully evaluate the performance of TFRC Veno, our simulation uses two different random loss models – exponential distribution and two-state distribution – in

Figure 4.1: Topology of simulation experiments.

the wireless links. In the exponential distribution, packet loss rate ranges from 0.0001 to 0.1. In the two-state distribution, the good state has packet loss rate of 0.00001 and period of 8 s, and the bad state has packet loss rate ranging from 0.0001 to 0.1 and period of 4 s. Furthermore, the good state has 95% probability to transit to the bad state after every period, so does the bad state.

• Throughput

Firstly we let 8 target (TFRC or MULTFRC or TFRC Venet) flows run in the experiment and calculate their average throughputs. Here the throughput is counted as the number of packets sent divided by the transmission time. Figure 4.2.a and 4.2.b plot the throughput comparisons of these three protocols – TFRC, MULTFRC, and TFRC Venet – under different types of random losses. Note that the packet loss rate in Figure 4.2.b corresponds to the bad state packet loss rate in the two-state distribution. As shown in these two figures, TFRC Venet has a similar throughput as MULTFRC: its throughput is a little lower than that of MULTFRC when random loss follows the exponential distribution while a little higher when random loss follows the two-state distribution. Furthermore, both TFRC Venet and MULTFRC has much better throughput than TFRC. In particular, the throughput improvement can be up to 300% over TFRC when the packet loss rate is 0.1.

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

4.2.a: Exponential distribution case.

4.2.b: Two-state distribution case.

Figure 4.2: Throughput comparisons under different random loss rates.

Secondly, we vary the number of target flows from 4 to 32 and study their performance. The random packet loss rate is set to 0.01. Observing Figure 4.3, our proposal also has a similar performance as MULTFRC under different numbers of connections. As the number of flows increases, which means the congestion is dominating the network, both TFRC Veno and MULTFRC performs increasingly similar to TFRC.

4.3.a: Exponential distribution case.

4.3.b: Two-state distribution case.

Figure 4.3: Throughput comparisons under multiple connections (packet loss rate is 0.01).

• Fairness

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

Fairness means the same kind of flows should share the total bandwidth fairly. Figure 4.4.a and Figure 4.4.b plot the sequence numbers of 16 MULTFRC flows and 16 TFRC Veno flows respectively, when they run over a network where random losses follow the exponential distribution and the loss rate is 0.01. Comparing these two figures, TFRC Veno flows share the bandwidth equally while the evolution of the flows among MULTFRC connections may have significant difference.

4.4.a: MULTFRC, $f=0.967$.

4.4.b: TFRC Veno, $f=0.998$.

Figure 4.4: Sequence number against time, 16 connections, random losses follow exponential distribution, packet loss rate is 0.01.

In order to have a better fairness comparison among TFRC, MULTFRC and TFRC Veno, Jain's Fairness Index f [81] is adopted here:

$$f = \left(\sum_{i=1}^n x_i \right)^2 / \left(n \sum_{i=1}^n x_i^2 \right) \quad (4.2)$$

where n is the number of connections, x_i is the throughput of the i th connection. The closer f is to 1, the more fairness the target flows enjoy. It should be pointed out that f is not a sensitive function. That is to say, a quite different fairness situation can only result in a little variation of f . For example, f of MULTFRC in Figure 4.4.a is only 0.03 less than that of TFRC Veno in Figure 4.4.b, but the gap

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

between the fastest flow and the slowest flow of MULTFRC is 6 times larger than that of TFRC VenO.

We set the number of target flows to 8, and study the fairness of the three protocols under different random packet losses. As seen in Figure 4.5, TFRC VenO has a good fairness as in TFRC (the fairness index is near 1). Both TFRC VenO and TFRC have better fairness than MULTFRC.

4.5.a: Exponential distribution case.

4.5.b: Two-state distribution case.

Figure 4.5: Fairness comparisons under different random loss rates.

Then we set the packet loss rate to 0.01, and vary the number of target flows from 4 to 32. As the number increases, TFRC VenO keeps a good fairness as in TFRC. However, the fairness of MULTFRC becomes increasingly poor. Figure 4.6 plots these results.

• TCP-friendliness

TCP-friendliness measures whether the target flows are aggressive or not when competing with TCP flows. It can be studied as follows: we first set up a certain number of TCP Sack flows over the link, and calculate their average throughput T_1 . Then we replace half of them with the target flows and recalculate the average

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

4.6.a: Exponential distribution case.

4.6.b: Two-state distribution case.

Figure 4.6: Fairness comparisons under multiple connections (packet loss rate is 0.01).

throughput of the remaining TCP Sack flows T_2 . If $\frac{T_2}{T_1}$ equals to 1, it means Sack flows are not affected by the target flows, and thus the target flows are totally friendly. The closer $\frac{T_2}{T_1}$ is to 1, the more friendly the target flows are.

4.7.a: Exponential distribution case.

4.7.b: Two-state distribution case.

Figure 4.7: TCP-friendliness comparisons under different random loss rates.

Figure 4.7 shows the results when there are 8 connections, and random packet loss ranges from 0.0001 to 0.1. Figure 4.8 plots the results under different numbers (ranging from 4 to 32) of connections, where the random packet loss rate is 0.01. Note that in all experiments, we always replace half of the TCP Sack connections

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

4.8.a: Exponential distribution case.

4.8.b: Two-state distribution case.

Figure 4.8: TCP-friendliness comparisons under multiple connections (packet loss rate is 0.01).

with target (TFRC or MULTFRC or TFRC VenO) connections. As shown in these figures, TFRC VenO flows are not harmful to the existing TCP Sack flows when random loss follows the exponential distribution ($\frac{T_2}{T_1}$ is always around 1), and are acceptable when random loss follows the two-state distribution ($\frac{T_2}{T_1}$ is close to 0.9). On the other hand, MULTFRC is always very aggressive in either cases (because $\frac{T_2}{T_1}$ is less than 0.6).

• Sending Rate Variation

As a transport protocol for streaming multimedia transmission, the sending rate of the target flows should not vary excessively. Here, we use the coefficient of variation (CoV) to measure this variation. CoV is defined as the standard deviation divided by the mean:

$$CoV = \frac{\sigma}{\mu}, \quad (4.3)$$

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i, \quad (4.4)$$

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2} \quad (4.5)$$

where, x_i is the i th sample, and N is the number of samples.

In the experiment, a certain number of the target flows run with the same number of TCP Sack flows. For each target flow, we collect the number of packets it sends in every 0.1 second as the samples (x_i), and use these samples to calculate its CoV after transmission. The smaller CoV is, the smoother the sending rate it has.

Figure 4.9 shows the average results when 4 target flows run with 4 TCP Sack flows, and Figure 4.10 plots the average results when the total numbers of flows (half are the target flows) increases from 4 to 32 where the packet loss rate is 0.01. The results demonstrate that MULTFRC has a smoother sending rate in most cases. However, TFRC Veno can also keep CoV at small values, which are almost the same as those of TFRC.

4.9.a: Exponential distribution case.

4.9.b: Two-state distribution case.

Figure 4.9: Sending rate variation comparisons under different random loss rates.

In summary, from the comparisons of throughput, fairness, TCP-friendliness, and sending rate smoothness we can conclude that, TFRC Veno is an efficient enhancement

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

4.10.a: Exponential distribution case.

4.10.b: Two-state distribution case.

Figure 4.10: Sending rate variation comparisons under multiple connections (packet loss rate is 0.01).

of TFRC since the significant throughput improvement in TFRC Veno does not sacrifice the other desirable features of TFRC such as fairness, TCP-friendliness, and sending rate smoothness.

4.1.2.2 Internet Experiments

We use the same link between NTU and HKU as described in Section 3.1.2.2. As shown in Figure 4.11, the TFRC/TFRC Veno client is at NTU and the TFRC/TFRC Veno server is at HKU.

At first, we report the single connection results. We select six time slots (1 hour period) over a day. In each time slot, we first let a TFRC flow run from the server to the client for 5 minutes, and then let a TFRC Veno flow run in the same direction for 5 minutes. After a 2-minute break, we repeat such test. Thus totally there are five tests in each time slot, and we calculate the average results of TFRC and TFRC Veno, respectively. Three measurements are calculated here: the number of packets sent, the receiving ratio, and the CoV. Note that the packet size is 500 bytes during transmission. We performed five days of testing from Monday to Friday. Here we only present the

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

Figure 4.11: Topology of the Internet experiments.

results of one day because the general trend of the results in each day is similar. As depicted in Table 4.1, TFRC Veno can obtain throughput improvements of up to 70% over TFRC, with the same receiving ratio.

Table 4.1: The Internet experiment results on single flow.

TFRC/TFRC Veno						
Time Slots	9:00~10:00	11:00~12:00	13:00~14:00	15:00~16:00	17:00~18:00	22:00~23:00
Packets Sent	2916.8/4128.6	2458.3/3249.7	2037.3/2981.3	2165.3/2640.0	3471.2/3507.4	1058.3/1812.7
Receiving Ratio	0.83/0.83	0.84/0.83	0.78/0.81	0.80/0.81	0.90/0.88	0.71/0.73
CoV	0.33/0.37	0.27/0.22	0.19/0.24	0.25/0.31	0.13/0.11	0.34/0.29

Table 4.2: The Internet experiment results on coexisting flows.

Time Slots	10:30~11:30	13:30~14:30	16:30~17:30	19:30~20:30	21:30~22:30
TFRC Veno/TCP Sack	1469.8/1132.3	1657.5/1253.3	2038.7/1202.5	2212.3/1357.8	1762.5/1224.5
TFRC/TCP Sack	1462.3/1340.0	1159.3/1179.8	1496.8/1343.3	1609.8/1525.3	1435.5/1349.0
TCP Sack/TCP Sack	1390.0/1021.8	1328.3/1347.5	1364.8/1404.8	1299.0/1312.8	1225.3/1458.8

We also test TFRC Veno's performance when coexisting with one competing TCP Sack flow. We select five time slots (1 hour period) over a day. In each time slot, we

sequentially let one TFRC Veno flow, one TFRC flow, and one TCP Sack flow run with an existing TCP Sack flow for 5 minutes. This test will be repeated four times in each time slot, and we calculate the average number of packets transmitted in each flow. Note that all flows have same packet sizes of 500 bytes and are transferred from the same server to the same client. Since the general trend of the results in each day of the entire five-day period is similar, we only present the results of one day. As shown in Table 4.2, the TFRC Veno flow does not cause any bias degradation to the competing TCP Sack flow, and obtains higher throughput while compared to the TFRC flow.

4.1.3 Summary

TFRC is designed to mainly provide TCP-friendly service in transport layer for streaming applications. In wireless environments TFRC suffers throughput degradation due to its embedded Reno equation. In this work, we replace the Reno equation by the Veno equation, which is derived from the throughput model of TCP Veno, to enhance TFRC performance. The comprehensive experiments show that our enhancement can improve TFRC performance significantly over wireless networks for multimedia transmission. As compared to another sender-modified enhancement MULTFRC, our proposal is more friendly to other network traffic when achieving the same high throughput.

4.2 Veno State Differentiator Based TFRC Enhancement

In this work, we study another possible way to improve TFRC performance – the Veno state differentiator based enhancement. Specifically, our enhancement applies the state differentiator of TCP Veno [20] to distinguish non-congestion losses and congestion losses. Non-congestion losses will make less contribution to the calculation of the sending rate than congestion losses do. Our simulation results show that, such Veno state differentiator

based modification can improve TFRC performance up to 70% over wireless networks. Furthermore, this proposal is an end-to-end enhancement, which does not require any supports from the intermediate nodes.

4.2.1 TFRC with Veno State Differentiator

As described in Section 2.1.2.1, the Veno state differentiator uses the measurement of $N = \frac{cwnd}{RTT} \times (RTT - BaseRTT)$ as an indication of whether the network is in congestive state or non-congestive state. Specifically, if $N \geq \beta$, Veno deduces the link is in congestive state and considers the packet loss is congestion loss. Otherwise if $N < \beta$, the link is in non-congestive state and the packet loss is non-congestion loss.

In our proposal, we let the TFRC receiver distinguish non-congestion losses from congestion losses based on the Veno state differentiator. So the receiver needs to know the measurement N , which can be obtained from the sender:

$$N = T_{current} \times (RTT - BaseRTT)$$

Here $T_{current} = \frac{cwnd}{RTT}$, which can be regarded as the current sending rate. The sender measures the $BaseRTT$ and RTT , and then calculates the value of N using $T_{current}$. After that, it sends N to the receiver.

When the receiver detected a new loss event i , it will calculate the loss interval $\bar{A}_i$ based on the current value of N . If $N \geq \beta$, it deduces such loss event is congestion loss event, and calculate the loss interval $\bar{A}_i$ as described in Section 2.2.1.2, i.e., $\bar{A}_i = A_i$. If $N < \beta$, however, it deduces the loss event is non-congestion loss event. To reduce the contribution of non-congestion loss event to the whole loss event rate p , we extend the current loss interval A_i by certain times α , i.e., $\bar{A}_i = \alpha A_i$. According to equation (2.10)–(2.13), the larger loss interval ($\alpha > 1$) results the smaller loss event rate p .

To determine the value of α , consider the following extreme scenario: if all loss events are non-congestion loss events, then

$$\bar{A}_i = \alpha A_i \quad \forall i \quad (4.6)$$

According to equation (2.10)–(2.13) we have

$$\bar{A}(1, n) = w \sum_{i=1}^n w_i \bar{A}_i = \alpha w \sum_{i=1}^n w_i A_i = \alpha A(1, n) \quad (4.7)$$

$$\bar{A}(0, n-1) = w \sum_{i=0}^{n-1} w_{i+1} \bar{A}_i = \alpha w \sum_{i=0}^{n-1} w_{i+1} A_i = \alpha A(0, n-1) \quad (4.8)$$

$$\bar{A} = \max(\bar{A}(1, n), \bar{A}(0, n-1)) = \alpha \max(A(1, n), A(0, n-1)) = \alpha A \quad (4.9)$$

and,

$$\bar{p} = \frac{1}{\bar{A}} = \frac{1}{\alpha A} = \frac{1}{\alpha} p \quad (4.10)$$

According to [36], the throughput of TCP Veno is $\sqrt{3}$ times of that of TCP Reno if all the losses are non-congestion losses and all the time-outs are ignored. Then we have:

$$\bar{T} = \frac{1}{RTT} \sqrt{\frac{3}{2b\bar{p}}} = \frac{1}{RTT} \sqrt{\frac{3\alpha}{2bp}} = \sqrt{3}T = \frac{1}{RTT} \frac{3}{\sqrt{2bp}} \quad (4.11)$$

From equation (4.11) we get:

$$\alpha = 3 \quad (4.12)$$

In summary, the loss interval $\bar{A}_i$ is calculated as follows:

$$\bar{A}_i = \begin{cases} A_i, & N \geq \beta \\ 3A_i, & N < \beta \end{cases}$$

4.2.2 Performance Evaluation

The evaluation of our enhancement (hereinafter, called “Enhancement”) is based on NS-2 simulation experiments. Different aspects of performance such as throughput, receiving

ratio, sending rate variation, fairness, and TCP-friendliness, have been extensively studied in the following section.

Note that in TCP Veno, β is normally set to be 3. However, after a complete evaluation on different values of β in the following simulation experiments, we find $\beta = 1$ is a better choice for our TFRC enhancement on achieving a good tradeoff between throughput and TCP-friendliness.

The topology of our experiment on NS-2 is shown in Figure 4.12.

Figure 4.12: Topology of NS-2 experiments.

The left side network has wired links with bandwidth of 10 Mbps, delay of 1 ms and packet buffer size of 50. The right side network has wireless links with bandwidth of 10 Mbps, delay of 1 ms and packet buffer size of 50. Random losses in wireless links follow exponential distribution and the packet loss rate ranges from 0.0001 to 0.1. The bottleneck link between these two networks is a wired link. The bandwidth is 5 Mbps and delay is 80 ms. Its packet buffer size is set to be 20. TFRC packets are transferred from the wired network to the wireless network. Packet size is 1000 bytes.

• Throughput

At first we let a single TFRC or Enhancement flow run over the network. Meanwhile, we set up three UDP connections with pareto distribution over the bottleneck

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

link as the background traffic. The packet size of UDP is 512 bytes. The burst time and idle time are both 100 ms. The rate of each connection is set to 500 Kbps. The experiment time is 300 s. Figure 4.13.a plots the throughput results of TFRC and Enhancements with different values of β (1, 2, and 3). As shown in the figure, different values of β bring a little difference in the throughput for each Enhancement. However, all of these three Enhancements improve much throughput (up to 70%) over the original TFRC.

4.13.a: Single flow.

4.13.b: Multiple flows, random loss rate is 0.01.

Figure 4.13: Throughput comparison.

Then we increase the number of flows from 4 to 64 and compare the throughputs of TFRC and Enhancement when random loss rate is 0.01. Figure 4.13.b observes that, as the number of connections increases, the network is dominated by congestion and Enhancement performs similarly as TFRC does finally.

• Receiving ratio

We also compare the receiving ratio between our enhancement and TFRC during the experiments in throughput before. Receiving ratio is defined as follows:

$$\text{Receiving Ratio} = \frac{\text{Number of packets received}}{\text{Number of packets sent}}$$

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

which indicates how many packets are lost during transmission.

4.14.a: Single flow.

4.14.b: Multiple flows, random loss rate is 0.01.

Figure 4.14: Receiving ratio comparison.

From Figure 4.14.a and 4.14.b we can see, Enhancement almost keeps the same receiving ratio as that of TFRC under single connection. However, as the number of connections increase, Enhancement becomes losing a few more packets than TFRC does. This is because of the accuracy of Veno's state differentiator. As stated in [85], Veno's state differentiator works well under light load networks, but will misinterpret some congestion losses as non-congestion losses when the network load is heavy. The larger the threshold β is, the more misinterpretations Veno's differentiator has. In this case, our proposal based on Veno's state differentiator will overestimate the sending rate and thus causes slightly more packet losses during transmission. Furthermore, the larger the threshold β is, the higher the overestimated sending rate is, and the more packet losses our enhancement has.

- **Sending rate variation**

We also use coefficient of variation (CoV) defined in 4.3 to measure the sending smoothness of our proposal.

Figure 4.15: CoV comparison.

In the experiment, at first 4 TFRC flows run with 4 TCP SACK flows for 300s. Then we replace 4 TFRC flows with Enhancement flows and redo the experiment. We collect the number of packets the target flow sends in every 0.1s as samples, and calculate its CoV. Figure 4.15 plots the average results. As we see, our proposal does not sacrifice the smoothness of the sending rate.

• Fairness

We continue to use the Jain's Fairness Index f to reflect the fairness:

$$f = \frac{\left(\sum_{i=1}^n x_i \right)^2}{n \sum_{i=1}^n x_i^2}$$

where, n is the number of connections, x_i is the throughput of the i th connection. The closer f is to 1, the more fairness the flow has. Here we let 4 to 64 Enhancement connections run over the bottleneck link. As shown in Figure 4.16, Enhancement

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

can keep good fairness under different packet loss rates, no matter which threshold β is used.

4.16.a: $\beta = 3$.4.16.b: $\beta = 2$.4.16.c: $\beta = 1$.Figure 4.16: Fairness of Enhancement with different values of β .

• TCP-friendliness

We first set up certain number (ranging from 4 to 64) of TCP SACK flows over the link, and calculate their average throughput T_1 . Then we replace half of them with Enhancement flows and recalculate the average throughput of the left TCP SACK flows T_2 . The closer $\frac{T_2}{T_1}$ is to 1, the more friendliness Enhancement flows have. As illustrated in Figure 4.17, as the threshold value β increases, Enhancement flows becomes more and more aggressive, especially over heavy load networks. The reason is the same as that described in receiving ratio – the overestimated throughput makes Enhancement aggressive. However, the friendliness of Enhancement when $\beta = 1$ is acceptable since $\frac{T_2}{T_1}$ is always around 85% even under heavy load.

4.2.3 Summary

This work modifies TFRC protocol based on the Veno state differentiator. Such differentiator is used here to distinguish congestion loss events and non-congestion loss events. Our proposal discounts the impact of non-congestion loss events by extending their loss

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

Figure 4.17: Friendliness of Enhancement with different values of β .

intervals by 3 times. Simulation results have shown the better performance our enhancement achieves. Meanwhile, we also study how the threshold β used in the VenO state differentiator can influence the performance of our proposal. After a complete evaluation, we find $\beta = 1$ is a better empirical value on achieving a good tradeoff between throughput and TCP-friendliness.

4.3 TFRC VenO Implementation

As an important work for TFRC VenO, we implemented it into Linux Kernel 2.6.12. This work is based on the Linux Kernel implementations of TFRC and DCCP [86][87] maintained by Arnaldo C. Melo and Ian McDonald *et al* [88]. Here DCCP can be seen as a framework to provide unreliable data transmission, and TFRC is one optional choice of congestion control algorithms in DCCP [89]. In the following part, we will first review the DCCP implementation in Linux Kernel, and then show how to modify the TFRC module in DCCP to incorporate our TFRC VenO algorithm. Note that the implementations of TFRC and DCCP are only the experimental versions in Linux Kernel 2.6.12.

4.3.1 DCCP in Linux Kernel

4.3.1.1 Structures

DCCP has its own protocol control block in the Kernel. Its structure in the protocol switch table is shown in Listing 4.1.

Listing 4.1: DCCP protocol control block

```
static struct inet_protosw dccp_v4_protosw = {
    .type      = SOCK_DCCP,
    .protocol   = IPPROTO_DCCP,
    .prot       = &dccp_v4_prot,
    .ops        = &inet_dccp_ops,
    .capability = -1,
    .no_check   = 0,
    .flags      = 0,
};
```

Here `SOCK_DCCP` is the type of the DCCP protocol, which is used when creating socket for DCCP, e.g., `socket(AF_INET, SOCK_DCCP, 0)`. The structure `dccp_v4_prot` defines all functions the DCCP protocol supports, such as connecting, sending, and receiving, as shown in Listing 4.2. When a system call from socket layer comes down, the Kernel can use this structure to find the corresponding functions. For example, in response to the `connect` system call, the Kernel will call `dccp_v4_connect` to do the remaining work.

Listing 4.2: DCCP protocol functions

```

struct proto dccp_v4_prot = {
    .name          = "DCCP",
    .owner         = THIS_MODULE,
    .close         = dccp_close,
    .connect       = dccp_v4_connect,
    .disconnect    = dccp_disconnect,
    .ioctl         = dccp_ioctl,
    .init          = dccp_v4_init_sock,
    .setsockopt    = dccp_setsockopt,
    .getsockopt    = dccp_getsockopt,
    .sendmsg       = dccp_sendmsg,
    .recvmsg       = dccp_recvmsg,
    .backlog_rcv   = dccp_v4_do_rcv,
    .hash          = dccp_v4_hash,
    .unhash        = dccp_v4_unhash,
    .accept        = inet_csk_accept,
    .get_port      = dccp_v4_get_port,
    .shutdown      = dccp_shutdown,
    .destroy       = dccp_v4_destroy_sock,
    .max_header    = MAX_DCCP_HEADER,
    .obj_size      = sizeof(struct dccp_sock),
    .rsk_prot      = &dccp_request_sock_ops,
    .orphan_count  = &dccp_orphan_count,
};

```

4.3.1.2 Functions

Figure 4.18 gives the overview of the DCCP functions during data transmission. The sender sends data to the receiver and the receiver feeds back with acknowledgements (ACKs). In the sender side, when the DCCP socket has packets to send, it calls `dccp_sendmsg`. Then `dccp_sendmsg` calls `ccid_hc_tx_send_packet` to send out the packet. The function of `ccid_hc_tx_send_packet` is to choose one of the congestion control algorithms (CCID 1, CCID 2, ...) to determine the sending rate of the outgoing packets. As mentioned before, TFRC is one of these congestion control algorithms, and its congestion control identification is CCID 3. In this example, the TFRC module is selected. To determine the sending rate of the data, the TFRC module needs the information in the ACKs from the receiver. When the sender receives these ACKs, `dccp_v4_rcv` is called to process them. After that, `ccid_hc_tx_packet_rcv` is called to denote which congestion control module needs these ACKs. In this example, the TFRC module (CCID 3) receives

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

them, and calculates the sending rate based on the Reno equation.

Figure 4.18: DCCP functions.

In the receiver side, `dccp_v4_rcv` is called when the data is received. Then `dccp_v4_rcv` calls `ccid_hc_rx_packet_rcv` to select the congestion control module to process the data. In this example, the TFRC module is chosen, and it calculates the packet loss rate and the receiving rate according to the data received so far. Besides sending the data to the socket layer, the TFRC module also calls `dccp_send_ack` to feed back the information such as the packet loss rate and the receiving rate to the sender in ACKs.

4.3.1.3 CCID 3: TFRC Module

DCCP in the Linux Kernel holds a pointer array to store its congestion control modules. As shown in Figure 4.19, `ccid[255]` is an array storing the pointers which point to the congestion control modules in DCCP. Listing 4.3 shows the structure of the TFRC module (CCID 3).

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

Figure 4.19: CCID pointer array.

Listing 4.3: TFRC protocol functions

```

static struct ccid ccid3 = {
    .ccid_id          = 3,
    .ccid_name        = "ccid3",
    .ccid_owner       = THIS_MODULE,
    .ccid_init        = ccid3_init,
    .ccid_exit        = ccid3_exit,
    .ccid_hc_tx_init  = ccid3_hc_tx_init,
    .ccid_hc_tx_exit  = ccid3_hc_tx_exit,
    .ccid_hc_tx_send_packet = ccid3_hc_tx_send_packet,
    .ccid_hc_tx_packet_sent = ccid3_hc_tx_packet_sent,
    .ccid_hc_tx_packet_recv = ccid3_hc_tx_packet_recv,
    .ccid_hc_tx_insert_options = ccid3_hc_tx_insert_options,
    .ccid_hc_tx_parse_options = ccid3_hc_tx_parse_options,
    .ccid_hc_rx_init  = ccid3_hc_rx_init,
    .ccid_hc_rx_exit  = ccid3_hc_rx_exit,
    .ccid_hc_rx_insert_options = ccid3_hc_rx_insert_options,
    .ccid_hc_rx_packet_recv = ccid3_hc_rx_packet_recv,
};

```

When `ccid_hc_*` functions are called, a pointer named `ccid` is used to point to the congestion control module needed. For example, if `ccid_hc_tx_packet_recv` is called, and it passes `ccid` to point to the TFRC module `ccid3`, then the function `ccid3_hc_tx_packet_recv` in the TFRC module is called to process the remaining work. In the following part, we only focus on the functions in the TFRC module, which are shown in Figure 4.20.

In the receiver side, `ccid3_hc_rx_packet_recv` is called in the TFRC module to process the data. Beside passing the data to the upper layer, it calls `ccid3_hc_rx_send_feedback` to

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

Figure 4.20: TFRC/TFRC Veno module.

calculate the packet loss rate and the receiving rate. After that, it calls `dccp_send_ack` to send these information to the sender in ACKs.

In the sender side, the function `ccid3_hc_tx_packet_rcv` processes the received ACKs. It updates the round-trip time (RTT) based on these ACKs, and calculates the new sending rate for the data by calling `ccid3_hc_tx_update_x`. It is known that TFRC calculates its sending rate mainly based on the Reno equation. In `ccid3_hc_tx_update_x`, this work is done by the function `ccid3_calc_x`. Besides the result from the Reno equation, `ccid3_hc_tx_update_x` also has the responsibility to check the result satisfying other constraints, e.g., the result should not exceed two times of the receiving rate reported by the receiver. The detailed constraints of the sending rate calculation can be referred to [10]. After getting the final sending rate, `ccid3_hc_tx_update_x` schedules the sending timer based on it. After that, `ccid_hc_tx_send_packet` uses this sending timer to send out the data.

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

It is worthy to note that the floating point type is not allowed in the Linux Kernel, thus the calculation of the Reno equation, i.e., `ccid3_calc_x`, must be carefully designed. The TFRC module in the Linux Kernel assumes $b = 1$ in the Reno equation (2.8). Considering $RTO = 4 \times RTT$, the Reno equation can be arranged as:

$$T_{calc} = \frac{S}{RTT \times f(p)} \quad (4.13)$$

where,

$$f(p) = \sqrt{\frac{2p}{3}} + 12\sqrt{\frac{3p}{8}}p(1 + 32p^2) \quad (4.14)$$

The TFRC module keeps a table `calcx_lookup[500][2]` to store the static values of $f(p)$, where `calcx_lookup[*][1]` stores the values of $f(p)$ when $p \in [0.0001, 0.05]$, and `calcx_lookup[*][0]` stores the values of $f(p)$ when $p \in [0.002, 1]$, as shown in Figure 4.21. Given a loss rate p , the TFRC module converts it to an index, and looks up the corresponding value of $f(p)$ in the table directly: if $p < 0.05$, then it looks up in `calcx_lookup[*][1]`; otherwise, it looks up in `calcx_lookup[*][0]`. To avoid the floating point type variables, the values in $f(p)$ are multiplied by 1,000,000 times in the Linux Kernel.

Note that, the calculation of $f(p)$ in `calcx_lookup[500][2]` is based on the real values of p , but not the variable p indicating the packet loss rate in the Linux Kernel, because this variable is also multiplied by 1,000,000 times. To convert the multiplied variable p to the real index for the table `calcx_lookup[500][2]`, the TFRC module perform following work shown in Listing 4.4.

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

calcx_lookup[500][2]		
	0	1
0	f(0.002)	f(0.0001)
	.	f(0.0002)
	.	
	.	
24	f(0.05)	
25	f(0.052)	.
	.	.
	.	.
	.	.
498	f(0.998)	f(0.049)
499	f(1)	f(0.05)

index (p<0.05) →

index (p>=0.05) →

Not used

Figure 4.21: Table for the values of $f(p)$.

Listing 4.4: Conversion from p to index

```

if (p < 50000)
    index = (p / 100) - 1;
else
    index = (p / 2000) - 1;
... ..
if (p >= 50000)
    f = calcx_lookup[index][0];
else
    f = calcx_lookup[index][1];

```

After getting the value of $f(p)$, the TFRC module can calculate the result of the Reno equation based on (4.13). Note that RTT is in millisecond in the Kernel, thus the value of S must be multiplied by 1,000,000,000 times to make the final result correct.

4.3.2 TFRC Veno Module

As stated in Section 4.1, an attractive feature of TFRC Veno is that it needs only a minor modification in the TFRC sender. As for the TFRC module in the Linux Kernel, we only need to add a function called `tfrc_veno_calcx` in `ccid3_hc_tx_update_x`, as shown in Figure 4.20. `tfrc_veno_calcx` takes the responsibility to calculate the sending rate based on the

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

Veno equation instead of the Reno equation.

The calculation of the Veno equation is similar to the method described in Section 4.3.1.3. However, the situation is more complicated in the Veno equation because the Veno parameter γ is added.

We also arrange the Veno equation (4.1) by considering $b = 1$ and $RTO = 4 \times RTT$:

$$T_{calc} = \frac{S}{RTT \times f(p)} \quad (4.15)$$

where,

$$f(p) = a(p)c1(\gamma) + b(p)c2(\gamma) \quad (4.16)$$

$$a(p) = \sqrt{2p}, \quad b(p) = 12\sqrt{\frac{p}{2}}p(1 + 32p^2) \quad (4.17)$$

$$c1(\gamma) = \sqrt{\frac{1-\gamma}{1+\gamma}}, \quad c2(\gamma) = \sqrt{1-\gamma^2} \quad (4.18)$$

That is to say, to obtain the result from $f(p)$, we must calculate $a(p)$, $b(p)$, $c1(\gamma)$, and $c2(\gamma)$ respectively. Thus we keep three static tables in the Kernel: `calcx.A[500][2]`, `calcx.B[500][2]`, and `calcx.C[451][2]`. Here `calcx.A[500][2]` and `calcx.B[500][2]` have the same structure as `calcx.lookup[500][2]` described in Section 4.3.1.3, which are used to map p to the corresponding $a(p)$ and $b(p)$ respectively. `calcx.C[451][2]` is a new structure for the Veno parameter $\gamma \in [0.5, 0.95]$, where `calcx.C[*][0]` stores the values of $c1(\gamma)$, and `calcx.C[*][1]` stores the values of $c2(\gamma)$. Note that all these values are multiplied by 1,000,000 times. Figure 4.22 gives the detailed information.

To avoid the floating point type value of γ , its value is multiplied by 1,000,000 times. As similar to the TFRC module, we also need to convert the variable p or γ to the index for the table `calcx.A[500][2]` or `calcx.B[500][2]` or `calcx.C[451][2]`. The conversion and calculation of $f(p)$ is shown in Listing 4.5.

CHAPTER 4. ANALYSIS AND ENHANCEMENTS OF END-TO-END CONGESTION CONTROLS FOR STREAMING OVER WIRELESS NETWORKS

calcx_C[451][2]		
	0	0
0	c1(0.5)	c2(0.5)
1	c1(0.501)	c2(0.501)
2	c1(0.502)	c2(0.502)
	⋮	⋮
449	c1(0.949)	c2(0.949)
450	c1(0.95)	c2(0.95)

index (c1) →

← index (c2)

Figure 4.22: Table for the values of $c1(\gamma)$ and $c2(\gamma)$.

Listing 4.5: Conversion from p and γ to index

```

index = (gama / 1000) - 500;
... ..
c1 = calcx_C[index][0]; //look up c1( $\gamma$ )
c2 = calcx_C[index][1]; //look up c2( $\gamma$ )

if (p < CALCX_SPLIT)
    index = (p / 100) - 1;
else
    index = (p / 2000) - 1;
... ..

/* look up a(p) and b(p) */
if (p >= 500) {
    a = calcx_A[index][0];
    b = calcx_B[index][0];
}
else {
    a = calcx_A[index][1];
    b = calcx_B[index][1];
}

f = (u64)a * (u64)c1 + (u64)b * (u64)c2; //calculate f(p)
do_div(f, 1000000); //make f(p) smaller for the following work

```

Finally, the TFRC Veno module can calculate the result of the Veno equation based

on equation (4.15).

4.3.3 Summary

The DCCP implementation in the Linux Kernel provides a framework for unreliable data transmission. The TFRC module is one of the congestion control algorithms (CCID 3) used in DCCP. Our TFRC Veno module only add a new calculation function in the TFRC module, to calculate the sending rate based on the Veno equation instead of the Reno equation. However, because the Linux Kernel does not allow the floating point type, these calculation functions must be carefully designed. Static tables are used in both TFRC and TFRC Veno modules to provide a method to calculate floating point values by integers.

Chapter 5

Conclusion and Future Work

Wireless networks bring performance degradation issue to end-to-end congestion control algorithms, either in bulk data transfer applications or in streaming applications. This thesis studies the wireless issue in these two different categories of end-to-end congestion control algorithms respectively. In the bulk data transfer category, the main object to study is TCP Veno, a novel TCP variant for wireless communications. In the streaming category, the main object to study is TFRC, a popular equation based rate control for multimedia applications. In the following part, we will summarize our research work on these two different protocols, and then give the possible directions for future work.

5.1 Summary

The research work on TCP Veno starts with the analytical model. We develop a throughput model for TCP Veno, to provide theoretical analysis on TCP Veno behavior over wireless networks. The derivation characterizes TCP Veno congestion control algorithm in both wire and wireless situations, considering the dynamic reduction on congestion window in TCP Veno for different types of packet losses. The derived equation, called “Veno equation”, is proved to be able to predict TCP Veno throughput accurately by extensive simulations and the Internet experiments. Here we choose PFTK method [11] to

CHAPTER 5. CONCLUSION AND FUTURE WORK

derive the Veno equation, because the equation obtained by PFTK method can be seamlessly imported into TFRC protocol. In this way, the Veno equation not only gives an in-depth view on TCP Veno about its behavior during network communications, but also provides a simple but efficient way to enhance TFRC protocol for streaming applications, i.e., TFRC Veno in this thesis.

The second issue of TCP Veno this thesis tries to explore is whether Veno has reached its optimal performance over wireless networks. Through the analysis on Veno behavior over different types of wireless networks, we find that the fixed reductions ($1/5$) of congestion window on random losses without concerning the network circumstance will constraint the possible improvement of Veno in throughput. This thesis studies two typical wireless networks that TCP Veno may experience – light load networks and high BDP networks. Light load networks mean that the network resources are not fully occupied and congestion losses are not pervasive during TCP communications. Wireless local area networks (wireless LAN) usually belong to this category. A new variable, named *congestion loss rate*, is introduced in this case. It helps Veno act more appropriately in response to random losses and grab more available bandwidth on the link, while there is no fairness or friendliness sacrificed. High BDP networks mean the BDP of the connection is larger than the buffer size on the bottleneck link. This definition covers very large scale in different types of networks, including the high speed networks and long delay networks. An approximated variable called *pBDP* is introduced to inform Veno situations of the network and refine its congestion window evolution. This alleviates the negative impact of random losses over the high BDP networks and improves Veno throughput.

TCP Veno provides an efficient solution for the end-to-end congestion control over wireless networks. Although it is designed for bulk data transfer, its core ideas, such as the Veno equation and the state differentiator, can also be integrated into congestion controls for streaming. This thesis proposes two enhancements for TFRC based on the Veno equation and the state differentiator respectively.

CHAPTER 5. CONCLUSION AND FUTURE WORK

The Veno equation based enhancement, also known as TFRC Veno, replaces the Reno equation with the Veno equation. Since the Veno equation models the behavior of TCP Veno, a wireless TCP variant, TFRC Veno can perform much better than the original TFRC does in wireless environments. The significant improvement obtained by TFRC Veno does not bias the fairness or TCP-friendliness of TFRC. This makes TFRC Veno outperforms MULTFRC, another TFRC enhancement which may cause intra- and inter-protocol fairness problems during transmission. Furthermore, TFRC Veno only involves a simple modification of TFRC on the sender side – replacement of the equation, and requires no changes on the receiver protocol stack or interventions of the intermediate network nodes. Thus it can be easily deployed in the current Internet.

The following table (Table 5.1) gives comparisons among TFRC, MULTFRC, and TFRC Veno over wireless networks based on our analysis and experiments.

Table 5.1: Comparisons of TFRC, MULTFRC, and TFRC Veno.

	TFRC	MULTFRC	TFRC Veno
Throughput	Bad	Good	Good
Fairness and TCP-friendliness	Good	Bad	Good
Sending rate smoothness	Good	Good	Good
Deployability	Good	Good	Good

Another way to enhance TFRC performance is to use the Veno state differentiator. This proposal keeps the Reno equation in TFRC intact, but uses the Veno state differentiator to distinguish congestion losses and random losses during transmission. By discounting the impact of those non-congestion losses on the throughput calculation, it can effectively alleviate the throughput degradation caused by wireless links, although

such improvement cannot reach as high as TFRC VenO does. It also maintains other merits of the original TFRC, such as sending rate smoothness, fairness, and TCP-friendliness. This enhancement is an end-to-end solution, which involves modifications of both sender and receiver sides.

5.2 Future Work

There are several interesting directions for the future work, based on the work we have done.

- **A More Accurate VenO Equation**

The derivation of the VenO equation in this thesis makes two simplified assumptions in the VenO algorithm: 1) we assume that the refined AI phase in VenO has negligible improvement in TCP throughput as compared to the refined MD phase; 2) we assume the congestion window values under random losses are uniformly distributed. While extensive experiments have proved that our VenO equation has a fairly good fit to the throughput of VenO in different environments, it is worthy to obtain a more complete equation for TCP VenO, which includes the refined AI algorithm of VenO and the accurate distribution of congestion window under random losses. We believe this future work will help to explore the behaviors of VenO in a more detailed way, and improve the performance of our TFRC VenO further.

- **A More Flexible VenO Algorithm for Diverse Wireless Networks**

This thesis has proposed two enhancements for TCP VenO to avoid its “blind” reduction of congestion window under random losses. Each of them uses an additional variable (*congestionlossrate* or *pBDP*) to indicate the network circumstance. This provides more detailed information about the network situations to VenO, and thus

CHAPTER 5. CONCLUSION AND FUTURE WORK

make it act more appropriately on dealing with random losses. If we regard the original Veno using the state differentiator as “one-factor” algorithm for wireless networks, then these two enhanced Veno algorithms that use not only the state differentiator but also additional network situation indication can be seen as “two-factor” algorithms. However, these proposals are only the preliminary results of this research direction, since either enhancement can only handle its own special network situation (light load or high BDP). In the future work, two issues should be considered: 1) how to indicate Veno with network circumstance more accurately; 2) how to make Veno enhancement suitable for not only one special type of networks, but also complicated heterogeneous networks. To realize these two objects, more useful variables should be studied to indicate the network situations. These variables may build up a “n-factor” Veno algorithm. Furthermore, these factors may contribute to the reduction of congestion window flexibly using a control method, e.g., fuzzy control [90][91][92][93].

- **Congestion Controls in Multicast of Streaming over Wireless Networks**

TFRC and TFRC Veno are both end-to-end congestion controls for streaming, which support single flow transmitted from one node to the other node in the network. Many multimedia applications such as IPTV or video conference, however, require flow delivery among several nodes at the same time in networks. This can be realized by multicast. As the reference to [94], multicast has two types of architectures – IP multicast [95] and overlay multicast [96][97]. IP multicast requires support from intermediate routers. Data originates from a sender, is replicated at intermediate routers if needed, and is forwarded until it reaches a destination. On the other hand, overlay multicast let end hosts perform all multicast functions, i.e., routers only perform unicast forwarding, and data is replicated at end hosts and

CHAPTER 5. CONCLUSION AND FUTURE WORK

delivered to other hosts using end-to-end transmission. No matter which architecture is employed, congestion controls are needed in multicast to avoid network collapse during transmission. Similar to the end-to-end congestion controls, these multicast congestion controls also encounter performance degradation issue over wireless networks.

Congestion control over IP multicast has been studied for many years and some solutions have been proposed, such as Representative [98], RLA [99], Golestani [100], MTCP [101], TFMCC [102], and SMCC [103]. Among these proposals, TFMCC and SMCC are two mechanisms based on TFRC rate control protocol. TFMCC is a single rate multicast congestion control, which means all the nodes in the multicast receive data with a same rate. On the other hand, SMCC is a multiple rate multicast congestion control, that is, different nodes may receive data with different rates. Since we have obtained a wireless enhancement of TFRC – TFRC VenO, it is possible to import TFRC VenO into TFMCC and SMCC to replace TFRC protocol. In fact, some preliminary results have been obtained in our experiments. They have shown the significant throughput improvement of both TFMCC and SMCC over wireless networks. In the future work, more experiments and more results are needed to completely analyze the performance of TFRC VenO in multicast congestion control functions.

Overlay multicast is an emerging technology in the Internet. Since it remains the end-to-end semantic during multicast transmission, traditional congestion controls such as TFRC should work well in this architecture. However, it is an interesting issue on how random losses impact the performance of the whole system, when wireless networks are introduced. It can be predicted that TFRC VenO should perform better than TFRC does in these wireless environments. More study and work are needed to prove that in the future work.

- **Veno II: A Uniform Platform of End-to-end Congestion Controls for Heterogenous Networks**

Bulk data transfer and streaming compose the daily applications of the Internet. As wireless communication technology has been making significant progress in the recent Internet, congestion control algorithms meet new issues over wireless networks. This thesis covers the study of wireless issue in end-to-end congestion controls for both bulk data transfer and streaming. The wireless enhancements proposed in this thesis, either for bulk data transfer or for streaming, originate from the idea of TCP Veno. In the future work, we will explore the possibility to integrate these two categories of wireless enhancements into a uniform platform, named “Veno II”, which is being developed by our whole team.

References

- [1] J. Postel, “Internet Protocol: DARPA Internet Program Protocol Specification,” *RFC 791*, September 1981.
- [2] J. Nagle, “Congestion Control in IP/TCP Internetworks,” *RFC 896*, January 1984.
- [3] S. Floyd and K. Fall, “Promoting the Use of End-to-end Congestion Control in the Internet,” *IEEE/ACM Transactions on Networking*, vol. 7, pp. 458–472, August 1999.
- [4] V. Jacobson, “Congestion Avoidance and Control,” in *ACM SIGCOMM '88*, pp. 314–329, August.
- [5] M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control,” *RFC 2581*, April 1999.
- [6] D. M. Chiu and R. Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” *Journal of Computer Networks and ISDN*, vol. 17, pp. 1–14, June 1989.
- [7] D. Bansal and H. Balakrishnan, “Binomial Congestion Control Algorithms,” in *IEEE INFOCOM '01*, pp. 631–640, April 2001.
- [8] D. Bansal, “Congestion Control for Streaming Video and Audio Applications,” *Master Thesis of Massachusetts Institute of Technology*, January 2001.
- [9] S. Floyd, M. Handley, J. Padhye, and J. Widmer, “Equation-based Congestion Control for Unicast Applications,” in *ACM SIGCOMM '00*, pp. 43–56, August 2000.

REFERENCES

- [10] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," *RFC 3448*, January 2003.
- [11] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *ACM SIGCOMM '98*, pp. 303–314, September 1998.
- [12] S. Floyd, "Congestion Control Principles," *RFC 2914*, September 2000.
- [13] T. V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 336–350, June 1997.
- [14] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 756–769, June 1997.
- [15] H. Balakrishnan, "Challenges to Reliable Data Transport over Heterogeneous Wireless Networks," *PhD Thesis of University of California, Berkeley*, August 1998.
- [16] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM Wireless Networks*, vol. 1, pp. 469–481, December 1995.
- [17] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *IEEE ICDCS '95*, pp. 136–143, May 1995.
- [18] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, vol. 24, pp. 10–23, December 1994.
- [19] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving Performance of TCP over Wireless Networks," in *IEEE ICDCS '97*, pp. 365–373, May 1997.
- [20] C. P. Fu and S. C. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 216–228, February 2003.

REFERENCES

- [21] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *ACM MobiCom '01*, pp. 287–297, July 2001.
- [22] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for Wireless IP Communications," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 747–756, May 2004.
- [23] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media," in *IEEE ICNP '99*, pp. 213–221, October 1999.
- [24] C. Parsa and J. J. Garcia-Luna-Aceves, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," in *IEEE ICNP '99*, pp. 213–221, October 1999.
- [25] A. Chockalingam, M. Zorzi, and V. Tralli, "Wireless TCP Performance with Link Layer FEC/ARQ," in *IEEE ICC '99*, pp. 1212–1216, June 1999.
- [26] S. M. ElRakabawy, A. Klemm, and C. Lindemann, "TCP with Adaptive Pacing for Multihop Wireless Networks," in *ACM MobiHoc '05*, pp. 288–299, May 2005.
- [27] D. Kliazovich and F. Granelli, "Cross-layer Congestion Control in Ad Hoc Wireless Networks," *ELSEVIER Ad Hoc Networks*, vol. 4, pp. 687–708, November 2006.
- [28] X. Wu, M. C. Chan, and A. L. Ananda, "TCP HandOff: A Practical TCP Enhancement for Heterogeneous Mobile Environments," in *IEEE ICC '07*, pp. 24–28, June 2007.
- [29] Q. X. Pang, S. C. Liew, C. P. Fu, W. Wang, and V. O. K. Li, "Performance study of TCP Veno over WLAN and RED router," in *IEEE GLOBECOM '03*, pp. 3237–3241, December 2003.
- [30] Linux Kernel, *available at <http://www.kernel.org/>*.
- [31] R. Chaudhary and L. Jacob, "ECN Based TCP-friendly Rate Control for Wireless Multimedia Streaming," in *IEEE ICCCN '03*, pp. 599–602, October 2003.

REFERENCES

- [32] L. Huang, U. Horn, F. Hartung, and M. Kampmann, "Proxy-based TCP-friendly Streaming over Mobile Networks," in *ACM WoWMoM '02*, pp. 17–24, September 2002.
- [33] J. Y. Pyun and Y. Kim and K. H. Jang and J. A. park and S. J. Ko, "Wireless Measurement Based Resource Allocation for QoS Provisioning over IEEE 802.11 Wireless LAN," *IEEE Transactions on Consumer Electronics*, vol. 49, pp. 614–620, August 2003.
- [34] V. Arya and T. Turletti, "Accurate and Explicit Differentiation of Wireless and Congestion Losses," in *IEEE ICDCSW '03*, pp. 877–882, May 2003.
- [35] M. Chen and A. Zakhor, "Rate Control for Streaming Video over Wireless," in *IEEE INFOCOM '04*, pp. 1181–1190, March 2004.
- [36] B. Zhou, C. P. Fu, D. M. Chiu, C. T. Lau, and L. H. Ngoh, "A Simple Throughput Model for TCP Veno," in *IEEE ICC '06*, pp. 5395–5400, June 2006.
- [37] B. Zhou, C. P. Fu, K. Zhang, C. T. Lau, and C. H. Foh, "An Enhancement of TCP Veno over Light-load Wireless Networks," *IEEE Communications Letters*, vol. 10, pp. 441–443, June 2006.
- [38] B. Zhou and C. P. Fu, "An Enhancement of TCP Veno over High Bandwidth-Delay Product Networks," in *IEEE LCN '06*, pp. 535–536, November 2006.
- [39] B. Zhou, C. P. Fu, and V. O. K. Li, "TFRC Veno: An Enhancement of TCP Friendly Rate Control over Wired/Wireless Networks," in *IEEE ICNP '07 (to appear)*, October 2007.
- [40] B. Zhou, C. P. Fu, C. T. Lau, and C. H. Foh, "An Enhancement of TFRC over Wireless Networks," in *IEEE WCNC '07*, March 2007.
- [41] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," *end2end-interest mailing list*, April 1990.
- [42] J. C. Hoe, "Improving The Start-up Behavior of A Congestion Control Scheme for TCP," in *ACM SIGCOMM '96*, pp. 270–280, August 1996.

REFERENCES

- [43] M. Aron and P. Druschel, “TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control,” *Technical Report (TR98-318), Department of Computer Science, Rice University*, 1998.
- [44] H. N. Wang, H. J. Xin, D. S. Reeves, and K. G. Shin, “A Simple Refinement of Slow-start of TCP Congestion Control,” in *IEEE ISCC '00*, pp. 98–105, July 2000.
- [45] N. N. Hu and P. Steenkiste, “Improving TCP Startup Performance Using Active Measurements: Algorithm and Evaluation,” in *IEEE ICNP '03*, pp. 107–118, November 2003.
- [46] M. Allman and V. Paxson, “On Estimating End-to-end Network Path Properties,” in *ACM SIGCOMM '99*, pp. 263–274, September 1999.
- [47] J. Hoe, “Startup Dynamics of TCP’s Congestion Control and Avoidance Schemes,” *Master Thesis of Massachusetts Institute of Technology*, 1995.
- [48] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,” *ACM Computer Communication Review*, vol. 26, pp. 5–21, July 1996.
- [49] S. Floyd and T. Henderson, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” *RFC 2582*, April 1999.
- [50] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgment Options,” *RFC 2018*, October 1996.
- [51] M. Mathis and J. Mahdavi, “Forward Acknowledgement: Refining TCP Congestion Control,” in *ACM SIGCOMM '96*, pp. 281–291, August 1996.
- [52] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm,” *ACM Computer Communication Review*, vol. 27, pp. 67–82, July 1997.
- [53] N. Cardwell, S. Savage, and T. Anderson, “Modeling TCP Latency,” in *IEEE INFOCOM '00*, pp. 1742–1751, March 2000.

REFERENCES

- [54] B. Sikdar, S. Kalyanaraman, and K. Vastola, “Analytic Models and Comparative Study of Latency and Steady-state Throughput of TCP Tahoe, Reno and SACK,” in *IEEE GLOBECOM '01*, pp. 1781–1787, November 2001.
- [55] M. Mellia, I. Stoica, and H. Zhang, “TCP Model for Short Lived Flows,” *IEEE Communications Letters*, vol. 6, pp. 85–87, February 2002.
- [56] M. Mitzenmacher and R. Rajaraman, “Towards More Complete Models of TCP Latency and Throughput,” *The Journal of Supercomputing*, vol. 20, no. 2, pp. 137–160, 2001.
- [57] E. Altman, K. Avrachenkov, and C. Barakat, “A Stochastic Model of TCP/IP with Stationary Random Losses,” in *ACM SIGCOMM '00*, pp. 231–242, August 2000.
- [58] M. Zorzi, A. Chockalingam, and R. Rao, “Throughput Analysis of TCP on Channels with Memory,” *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 1289–1300, July 2000.
- [59] V. Misra, W. B. Gong, and D. F. Towsley, “Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED,” in *ACM SIGCOMM '00*, pp. 151–160, August 2000.
- [60] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.
- [61] S. Low, “A Duality Model of TCP and Queue Management Algorithms,” *IEEE/ACM Transactions on Networking*, vol. 11, pp. 525–536, August 2003.
- [62] L. Brakmo, S. O'Malley, and L. Peterson, “TCP Vegas: New Techniques for Congestion Detection and Avoidance,” in *ACM SIGCOMM '94*, pp. 24–35, August 1994.
- [63] J. Ahn, P. Danzig, Z. Liu, and L. Yan, “Evaluation of TCP Vegas: Emulation and Experiment,” *ACM Computer Communication Review*, vol. 25, pp. 185–95, October 1995.

REFERENCES

- [64] J. Mo, R. J. La, V. Anantharam, and J. C. Walrand, "Analysis and Comparison of TCP Reno and Vegas," in *IEEE INFOCOM '99*, pp. 1556–1563, March 1999.
- [65] K. Zhang and C. P. Fu, "The Performance Study of TCP Veno Under Different Recovery Schemes," in *APCC '06*, pp. 1–5, August 2006.
- [66] K. K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," *RFC 3168*, September 2001.
- [67] L. A. Grieco and S. Mascolo, "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control," *ACM Computer Communication Review*, vol. 34, pp. 25–38, April 2004.
- [68] R. Wang, K. Yamada, M. Y. Sanadidi, and M. Gerla, "TCP with Sender-side Intelligence to Handle Dynamic, Large, Leaky Pipes," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 235–248, February 2005.
- [69] D. Sisalem and H. Schulzrinne, "The Loss-Delay Based Adjustment Algorithm: A TCP-friendly Adaptation Scheme," in *ACM NOSSDAV '98*, July 1998.
- [70] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," in *IEEE INFOCOM '99*, pp. 1337–1345, March 1999.
- [71] W. T. Tan and A. Zakhor, "Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol," *IEEE Transactions on Multimedia*, vol. 1, pp. 172–186, May 1999.
- [72] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A Model Based TCP-friendly Rate Control Protocol," in *ACM NOSSDAV '99*, June 1999.
- [73] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers – Flow Control for Multimedia Streaming," *Technical Report, Department of Computer Science, North Carolina State University*, April 2000.

REFERENCES

- [74] J. Crowcroft and P. Oechslin, “Differentiated End-to-end Internet Services Using A Weighted Proportional Fair Sharing TCP,” *ACM Computer Communications Review*, vol. 28, pp. 53–67, July 1998.
- [75] M. Chen and A. Zakhor, “AIO-TFRC: A Light-weighted Rate Control Scheme for Streaming over Wireless,” in *IEEE WirelessCom '05*, pp. 1124–1129, June 2005.
- [76] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala, “Improving Simulation for Network Research,” *Technical Report 99-702, University of Southern California, Los Angeles*, March 1999.
- [77] The Network Simulator, *available at <http://www.isi.edu/nsnam/ns/>*.
- [78] NS-2 Manual, *available at <http://www.isi.edu/nsnam/ns/doc/ns-doc.pdf>*.
- [79] C. P. Fu, W. Lu, and B. S. Lee, “TCP VenO Revisited,” in *IEEE GLOBECOM '03*, pp. 3253–3258, December 2003.
- [80] C. L. Zhang, C. P. Fu, M. T. Yap, C. H. Foh, K. K. Wong, C. T. Lau, and M. K. Lai, “Dynamics Comparison of TCP VenO and Reno,” in *IEEE GLOBECOM '04*, pp. 1329–1333, December 2004.
- [81] R. Jain, *The Art of Computer Systems Performance Analysis*. Wiley, New York, 1991.
- [82] Dummynet, *available at http://info.iet.unipi.it/~luigi/ip_dummynet*.
- [83] S. Floyd, “HighSpeed TCP for Large Congestion Windows,” *RFC 3649*, December 2003.
- [84] T. Kelly, “Scalable TCP: Improving Performance in Highspeed Wide Area Networks,” *ACM Computer Communication Review*, vol. 33, pp. 83–91, April 2003.
- [85] Z. X. Zou, B. S. Lee, and C. P. Fu, “Packet loss and congestion state in TCP VenO,” in *IEEE ICON '04*, pp. 731–735, November 2004.

REFERENCES

- [86] E. Kohler, M. Handley, and S. Floyd, “Designing DCCP: Congestion Control Without Reliability,” in *ACM SIGCOMM ’06*, pp. 27–38, September 2006.
- [87] E. Kohler, M. Handley, and S. Floyd, “Datagram Congestion Control Protocol (DCCP),” *RFC 4340*, March 2006.
- [88] DCCP implementation for Linux, available at <http://linux-net.osdl.org/index.php/DCCP>.
- [89] E. Kohler, M. Handley, and S. Floyd, “Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC),” *RFC 4342*, March 2006.
- [90] L. A. Zadeh, “Fuzzy Sets,” *Informat. Control*, vol. 8, pp. 338–353, 1965.
- [91] L. A. Zadeh, “Outline of a New Approach to the Analysis of Complex Systems and Decision Processes,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, no. 1, pp. 28–44, 1973.
- [92] E. Mamdani and S. Assilian, “An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller,” *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.
- [93] E. Mamdani, “Advances in the Linguistic Synthesis of Fuzzy Controllers,” *International Journal of Man-Machine Studies*, vol. 8, no. 6, pp. 669–678, 1976.
- [94] A. Ganjam and H. Zhang, “Internet Multicast Video Delivery,” *Proceedings of the IEEE*, vol. 93, pp. 159–170, January 2005.
- [95] S. Deering, “Multicast Routing in Internetworks and Extended LANs,” in *ACM SIGCOMM ’88*, pp. 55–64, August 1988.
- [96] Y. H. Chu, S. G. Rao, and H. Zhang, “A Case for End System Multicast,” in *ACM SIGMETRICS ’00*, pp. 1–12, June 2000.
- [97] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O’Toole, “Overcast: Reliable Multicasting with an Overlay Network,” in *USENIX OSDI ’00*, pp. 197–212, October 2000.

REFERENCES

- [98] D. DeLucia and K. Obraczka, “Multicast Feedback Suppression Using Representatives,” in *IEEE INFOCOM '97*, pp. 463–470, April 1997.
- [99] H. A. Wang and M. Schwartz, “Achieving Bounded Fairness for Multicast and TCP Traffic in the Internet,” in *ACM SIGCOMM '98*, pp. 81–92, September 1998.
- [100] S. J. Golestani and K. K. Sabnani, “Fundamental Observations on Multicast Congestion Control in the Internet,” in *IEEE INFOCOM '99*, pp. 990–1000, March 1999.
- [101] I. Rhee, N. Balaguru, and G. N. Rouskas, “MTCP: Scalable TCP-like Congestion Control for Reliable Multicast,” in *IEEE INFOCOM '99*, pp. 1265–1273, March 1999.
- [102] J. Widmer and M. Handley, “Extending Equation-based Congestion Control to Multicast Applications,” in *ACM SIGCOMM '01*, pp. 275–285, August 2001.
- [103] G. I. Kwon and J. W. Byers, “Leveraging Single Rate Schemes in Multiple Rate Multicast Congestion Control Design,” *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 1975–1986, December 2004.

Publications

- [1] B. Zhou, C. P. Fu, and V. O. K. Li, "TFRC VenO: An Enhancement of TCP Friendly Rate Control over Wired/Wireless Networks," *IEEE International Conference on Network Protocols (ICNP)*, October 2007.
- [2] C. P. Fu, B. Zhou, and J. L. Zhang, "Modeling TCP VenO over Wired/Wireless Networks," *IEEE Communications Letters*, Vol. 11, No. 9, 2007.
- [3] B. Zhou, C. P. Fu, C. T. Lau, and C. H. Foh, "An Enhancement of TFRC over Wireless Networks," *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2007.
- [4] B. Zhou, and C. P. Fu, "An Enhancement of TCP VenO over High Bandwidth-Delay Product Networks," *IEEE Conference on Local Computer Networks (LCN)*, November 2006.
- [5] B. Zhou, C. P. Fu, K. Zhang, C. T. Lau, and C. H. Foh, "An Enhancement of TCP VenO over Light-Load Wireless Networks," *IEEE Communications Letters*, Vol. 10 No. 6, 2006.
- [6] B. Zhou, L. H. Ngoh, B. S. Lee, and C. P. Fu, "HDA: A Hierarchical Data Aggregation Scheme for Sensor Networks," *ELSEVIER Computer Communications*, Vol. 29 Issue 9, 2006.
- [7] B. Zhou, C. P. Fu, D. M. Chiu, C. T. Lau, and L. H. Ngoh, "A Simple Throughput Model for TCP VenO," *IEEE International Conference on Communications (ICC)*, June 2006.

REFERENCES

- [8] B. Zhou, L. H. Ngoh, B. S. Lee, and C. P. Fu, “A Hierarchical Scheme for Data Aggregation in Sensor Network,” *IEEE International Conference on Networks (ICON)*, November 2004.