UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

School of Electronics and Computer Science

Enhancing TCP Delivery over Wireless Networks

by

Kai-Wen Lien

Thesis for the degree of Doctor of Philosophy

April 2009

Abstract

Wireless communication has become a significant life style in the daily use. The wireless communication can be used to extend the service of wired communication. Based on the idea of simplicity, Transmission Control Protocol (TCP) has been used widely over wired network. When network applications take place using wireless link, TCP is still useful because partial wired connection might be necessary. Although wired and wireless communication do share something in common, they have distinct features. Therefore, TCP needs to be adjusted to fit in the wireless environment.

This research aims to enhance wireless TCP performance. In order to study network protocols and behaviours, network simulators are often used for researchers to configure and monitor the network factors and system states. Unfortunately, most network simulators cannot demonstrate what the real network does. They are applications. In this research, a network simulator based on the real Linux TCP/IP stacks is proposed. This simulator is able to not only simulate the wired network, but also allow users to extend its structure for live wireless emulation. By means of simulator and emulator, researchers can understand and configure detail factors for further experiments. Eventually, a new wireless TCP enhancement is proposed.

In this research, some contributions are delivered. Firstly, a network simulator based on Linux TCP stacks is implemented. Secondly, a wireless emulator and test environment are built, so the wireless factors can be configured and performance can be monitored. Thirdly, a wireless TCP mechanism, TCP NewZag, is proposed. Finally, several experiments to show the value of NewZag are reported in the thesis.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the various people who provided me with helpful assistance. Without their care and thoughtfulness, I cannot make it.

First, I would like to thank my supervisor, Dr. Jeff Reeve. He showed me how to be a great academician and directed my research. I cannot do it without his help.

Second, I would like to thank the examiners, Dr. Vasilis Friderikos and Professor Ed Zaluska, of my oral defence. They gave me the suggestion of thesis revision.

Third, I would like to thank all my colleagues in the research group of Communications Research Group in University of Southampton. We have been working together. It has been one of the best memory in my life.

Finally, I would like to thank Dr. S. Y. Wang, who gave me detail descriptions of his simulator under BSD UNIX. I also want to thank to the members of the Linux mailing list for helping me with the Linux network technology when I had problems.

Most important, to my parents, my wife Sharon, my daughter Sandra, my mother in law and all my family.

Kai-Wen Lien University of Southampton
April, 2009

Acronyms

1G	First Generation Mobile systems
2G	Second Generation Mobile systems
3G	Third Generation Mobile systems
ABE	Available Bandwidth Estimation
ACK	Acknowledgment
ARQ	Automatic Repeat Request
AP	Access Point
BS	Base Station
BDP	Bandwidth Delay Product
BER	Bit Error Rate
BSD UNIX	Berkeley Software Distribution UNIX
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
cwnd	Congestion Window
EBSN	Explicit Bad State Notification
ECN	Explicit Congestion Notification
ERE	Eligible Rate Estimate
ETSI	European Telecommunications Standards Institute
EWMA	Exponential Weighted Moving Average
FH	Fixed Host
FDMA	Frequency Division Multiple Access
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
GUI	Graphic User Interface
IBSS	Independent Basic Service Set
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPSEC	IP Security Protocol
ISDN	Integrated Services Digital Network
I-TCP	Indirect-TCP
LDA	Loss Discrimination Algorithm
MSR	Mobile Support Router

MAC	Media Access Control
MH	Mobile Host
MHP	Mobile Host Protocol
MIMO	Multiple-input Multiple-output
MSS	Maximin Segment Size
MTU	Maximum Transmission Unit
NCPLD	Non-Congestion Packet Loss Detection
OSI	Open Systems Interconnection
P2P	Point-to-Point
PDA	Personal Digital Assistant
QoS	Quality of Service
RF	Radio Frequency
RED	Random Early Detection
RFC	Request For Comments
RFID	Radio Frequency Identification
RLC	Radio Link Control
ITU	International Telecommunication Union
RTO	Retransmission Timeout
RTT	Round Trip Time
SACK	Selective Acknowledgment
SFQ	Stochastic Fairness Queueing
SRP	Selective Repeat Protocol
ssthresh	Slow-start Threshold
PMP	Point-to-Multi-Point
SS	Subscriber Station
TCP	Transmission Control Protocol
TCPW	TCP Westwood
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
VOIP	Voice over IP
WECN	Wireless Explicit Congestion Notification
WiMAX	Wireless Metropolitan Area Networks
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Networks
WTP	Wireless Transaction Protocol
WWAN	Wireless Wide Area Network

Chapter 1

Introduction

Wireless communication technologies have been making significant progress in the recently years. The wireless network will play an important role in network access and Internet service. Wireless networks, such as cellular networks, wireless local area networks (WLANs), and wireless home networks, are usually interconnected with wired backbone networks. In this way, when a mobile device requests data transmission from a fixed host or a server, the transmission path generally crosses both wireless and wired network topologies. A network topology, which uses multiple network layer protocols and transmission materials, is described as a heterogeneous network or hybrid network.

Current network protocols are primarily optimized for a wired network and still face many challenges in serving the hybrid network. Improving current protocols to enhance network performance over heterogeneous network has become a critical task. But, protocol adjustments may render some current network devices redundant. Hardware costs may increase indirect effects on the development of new protocols. Besides, new protocols must also be backwards compatible. Backward compatibility refers to whether a newly introduced proposal is compatible with legacy protocol in the sense that it does not cause any detrimental effects to it, so the effects of network variance can be minimized.

Network applications over the heterogeneous network are generally served on top of the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP is one of the most widely used transport layer protocols in Internet. It is a connection-oriented protocol that implements many mechanisms to keep stable transmission. But, most TCP mechanisms, such as congestion control mechanism, was initially designed in use of the wired network. The congestion control mechanism performs well in the wired network and supports stable transmission when TCP detects a congestive network. But, it suffers from extra challenges when the TCP transmission has been extended to the wireless network. In the wired network, if the packet loss is detected, the TCP congestion control mechanism assumes that the loss is an indicator of network congestion. It will reduce congestion window (*cwnd*) to slow down the transmission. In the wireless network, a packet loss may occur as a result of noise, link error or reasons other than the network congestion. If TCP treats all packet losses as an indicator of congestive events and blindly reduces *cwnd*, it could decrease the TCP performance in the wireless network or the heterogeneous network. Hence, to enhance TCP performance over the heterogeneous network is the primary concept in this research.

1.1 Overview of Wireless Networks

Current wireless networks are classified in five types. Wireless Personal Area Networks (WPANs), Wireless Local Area Networks (WLANs), Wireless Metropolitan Area Networks (WMANs), Wireless Wide Area Networks (WWANs) and Wireless Sensor Networks (WSNs). WPANs provide low-power, short-range connectivity between mobile devices and the Internet. WLANs provide wideband local access. Unlike WPANs, WLANs provide continuous coverage for devices in the network. As devices might roam freely within the coverage areas, these coverage areas remain fixed. WLANs utilize electromagnetic waves to transfer data between equipment in a limited area. WMANs implement wireless technologies into a large area, such as a city. It is possible that WMANs can replace current last-mile technologies (e.g., xDSL). WWANs generally use digital cellular phone networks to enable laptops and Personal Digital Devices (PDAs) to access the Internet. Unlike WLANs, which offer limited user mobility and, instead, are generally used to enable the mobility of the entire network, WWANs facilitate connectivity for mobile users. WSNs consist of spatially distributed autonomous devices using sensors to monitor physical or environmental conditions, such as temperature, sound, and pressure, at different locations [1, 2]. Radio Frequency Identification (RFID) is another kind of wireless technology, which is not involved in the above classifications. This section describes different wireless technologies.

1.1.1 Wireless Personal Area Networks

A Wireless Personal Area Network (WPAN) is a personal area network for interconnecting devices centred around an individual person's workspace. Bluetooth [3, 4] is a well-known implementation of a WPAN, and uses the IEEE 802.15 standard. Bluetooth was designed as a low cost and low power radio technology, distinct from IEEE 802.11x [5, 6] WLAN technology, and will be described in a following sub-section. It is particularly suitable for short range personal networks. The main features of Bluetooth are:

1. Up to 1 Mbps transfer speeds.
2. 10-100 metre data transfer range.
3. Supports point-to-point connections without cables between mobile devices.

4. Supports point to multi-point connections to connect ad-hoc local wireless networks.

Recently, Bluetooth technology has been implemented in the rising Voice over IP (VOIP) scene, in which Bluetooth headsets can be used as wireless extensions to the computer audio system. For general home or office users, the VOIP has became more popular and convenient than wired phone lines, so Bluetooth could be used in cordless handsets, with a base station connected to the Internet. The next version of Bluetooth will allow Bluetooth to transfer data up to 480 Mbps, while building on the very low-power idle modes.

1.1.2 Wireless Wide Area Networks

A Wireless Wide Area Networks (WWAN) generally uses digital cellular phone networks to enable mobile devices to access the Internet, allowing users to maintain access to Internet information while away from the computer server or when traveling outside. Currently, the digital phone system is used in cellular network technology. The cellular network separates its territory into numbers of small areas called cells. Numbers of cells are controlled by one base station (BS) which is responsible for mobile communication and control. As this kind of network resembles like a honeycomb, it is called a cellular type mobile communication system and the corresponding network is called a cellular type network or cellular network. A cellular network has to meet certain criteria including [7]:

- Good subjective speech quality.
- Low terminal and service cost.
- Support for international roaming.
- Ability to support mobile terminals.
- Support for a range of new services and facilities.
- Spectral efficiency.
- Integrated Services Digital Network (ISDN) [8] compatibility.

A cellular network requires voice-oriented and data-oriented technologies. Global System for Mobile Communication (GSM) [9] is a renowned voice-oriented technology system. The data-oriented technology integrated with voice-oriented networks uses the same air-interface as those that have their own air-interface. The General Packet Radio Service (GPRS) [10] is an example of a data-oriented technology. The stages of development of cellular networks is introduced next.

- The First Generation Mobile (1G) systems were based on analogue signaling. Analogue systems were primarily based on circuit-switched technology, which was designed for voice transmission, not data delivery. The main drawbacks of 1G systems were low service quality, long call setup time and inefficient use of bandwidth. Besides, 1G systems were susceptible to interference and supported only insecure transmission.

- The Second Generation Mobile (2G) systems used digital modulation techniques and call processing methods. Most 2G systems combined Time Division Multiple Access (TDMA) and Frequency Division Multiple Access (FDMA) techniques to increase the number of channels. The Global System for Mobile Communication (GSM) systems was the most popular 2G system worldwide. In 1989, GSM responsibility was transferred to the European Telecommunication Standards Institute (ETSI). Nowadays, the acronym GSM is used to mean Global System for Mobile Communications. Compared with the 1G systems, the 2G systems provided better service quality and used the given bandwidth more efficiently. It supported both data, speech and image services. 2G systems combine advanced encryption mechanisms for data protection. The main drawbacks of 2G systems are low data transmission rates and is unsuitability for co-operation with current Internet systems.

- The General Packet Radio Service (GPRS)came between 2G and 3G. It applies packet radio principles to transfer data between GSM mobile stations and external packet data networks. GPRS supports X.25 [11], IPv4 [12] and IPv6 [13] networks and offers data rates up to 150Kbit/s. Compared with GSM, GPRS usage costs are relative to the amount of data transmitted rather than the duration of connection. This is suitable for applications with bursty traffic, such as web browsing or Email. GPRS was an important system to evolve 2G systems towards 3G systems. It offers packet switching to deliver general data and circuit switching to transfer voice data.

- The Third Generation Mobile (3G) systems provide high speed transmission of both voice and data. 3G systems have the ability to unify existing cellular standards, such as GSM and TDMA. Thus, systems integrate all kind of services, including speech, data, audio, video and facsimile. They provide a much better quality of service (QoS) than the 2G systems and use much smaller call set-up delay. The goal of 3G systems is to provide better service quality at low cost and shorter call set-up times. The integration of digital multimedia into mobile systems is another main development of 3G systems.

International Telecommunication Union (ITU) estimates that the worldwide mobile cellular subscriber base (2G, 3G) will reach the 4 billion mark before the end of 2008, displayed in Figure 1.1. The growth of mobile cellular subscribers has averaged 24 per cent between 2000 and 2008. While in 2000, mobile penetration stood at just 12 per cent, it surpassed the 50 per cent mark by early this year, and is expected to reach about 61 per cent by the end

2008. The surge in the mobile cellular subscriber base is mainly attributable to development in some of the world's largest markets. Brazil, Russia, India, and China are expected to account for over 1.3 billion mobile subscribers by the end of 2008 [14].

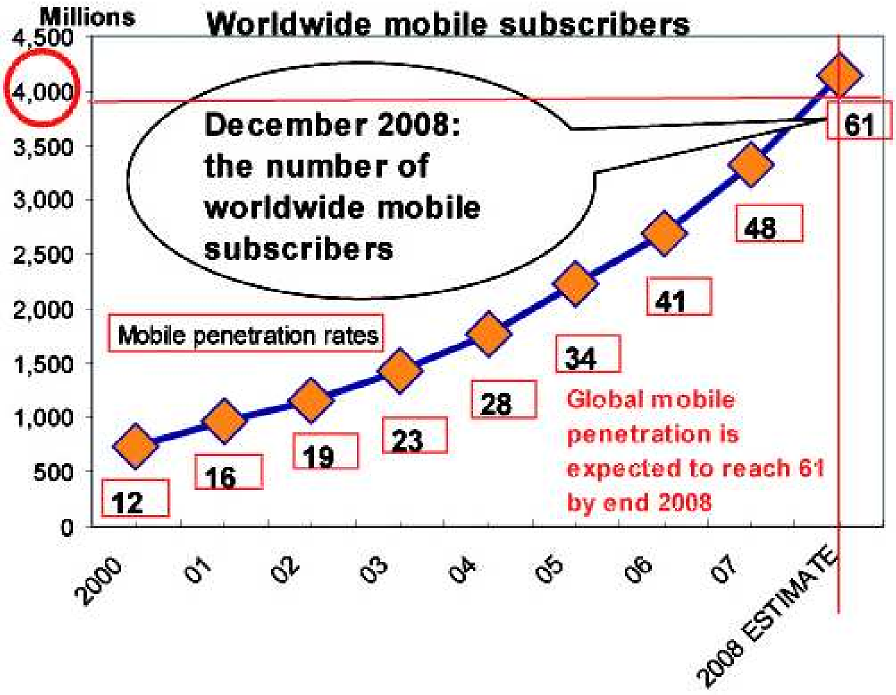


Figure 1.1: Worldwide mobile cellular subscribers. Source: ITU World Telecommunication/ICT Indicators (WTI) database.

1.1.3 Wireless Local Area Networks

A Wireless Local Area Network (WLAN) is a type of network that uses high-frequency radio waves rather than wires to communicate between devices in local areas. There are two types of WLANs, independent WLANs (or peer-to-peer) and infrastructure WLANs. An independent WLAN simply connects a set of computers with wireless adapters. Computers can set up an independent network any time when two or more wireless adapters are within range of each other. In general, these on-demand networks require no administration or pre-configuration. Access points (APs) are used to extend the signal by acting as a repeater, so the transmission distance between wireless devices is extended. A mobile ad-hoc Network [15] (also named peer-to-peer or Independent Basic Service Set, or IBSS) is a renowned specification for independent WLANs.

A WLAN structure is widely implemented today, in which a wireless network is linked to a wired network. In an infrastructure WLAN, the wireless network is connected to a wired network through access points (APs). APs possess both Ethernet links and antennae to send signals. In a WLAN, equipment can move within and between coverage areas without experiencing disruption in connectivity as long as they stay within range of an AP at all times.

IEEE 802.11x [5, 6] is a well-known standard implemented in WLANs. It was designed by the Institute of Electrical and Electronic Engineers (IEEE). The IEEE 802.11 standard is operated over the radio frequency (RF) band around 2.4GHz and provides for data rates between 1Mbps and 2Mbps. The IEEE 802.11a and IEEE 802.11b standards are defined at bands of 5.8GHz and 2.4GHz, respectively. The IEEE 802.11 standard places specifications on the parameters of both the physical and medium access control (MAC) layers of the network. The physical layer handles the transmission of data between nodes. The MAC layer is responsible for maintaining order in the use of a shared medium. The 802.11 standard specifies a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol. In this protocol, when a node prepares to transmit a packet, it first listens to the physical layer to ensure that no other node is transmitting. If the channel is clear, it sends the packet. Otherwise, it chooses a random "backoff factor", which determines a waiting time until it is allowed to transmit its packet.

Currently there are three major IEEE 802.11x wireless-networking standards.

1. 802.11b is the corporate specification and has a suitably wide range for use in big office spaces. Its maximum data transmission speed can be up to 11Mbps.

2. 802.11a offers bigger bandwidth and fewer interference problems but a shorter range. Currently, some manufacturers are modifying their equipment to handle 22 Mbps or more using this standard.

3. 802.11g is a new standard, with up to 54Mbps maximum data transmission speed. It is an extension of the 802.11b standard.

4. 802.11n builds on previous 802.11 standards by adding multiple-input multiple-output (MIMO) and Channel-bonding/40 MHz operation to the physical layer, and frame aggregation to the MAC layer. It supports a PHY rate of 300 Mbps.

A problem of WLAN is overcrowding of the bandwidth. If there are too many people or businesses using WLANs in the same area, it can overcrowd the frequency band that they are transmitting on.

1.1.4 Wireless Metropolitan Area Networks

The IEEE 802.16 standards [16, 17] or Worldwide Interoperability for Microwave Access (WiMAX) aims to prepare formal specifications for the global deployment of broadband Wireless Metropolitan Area Networks (WMANs). WiMAX is a broadband wireless system which offers packet switched services for fixed, portable and mobile accesses. A WiMAX base station can offer a greater coverage area at around five miles with line-of-sight (LOS) transmission within a bandwidth of up to 70 Mbps.

WiMAX defines two operational modes: a Point-to-Multi-Point (PMP) mode and a mesh mode. The PMP mode defines one-hop communication between a base station and a subscriber station (SS). SS refers to a generalized equipment set providing connectivity between subscriber equipment and a base station in the mobile wireless network. The PMP mode is designed to replace current last-mile technologies, such as xDSL.

In the mesh mode, SSs are called mesh SS (MSS) and BS is called mesh BS (MBS). In mesh mode, SSs do not need to be directly connected to the MBS, but can be connected to nearby SSs. The main difference between PMP mode and mesh mode is that traffic in the PMP mode only occurs between BS and subscriber stations (SSs). In mesh mode, traffic can be routed through other SSs.

WiMAX can offer a large area wireless coverage and high speed wireless transmission. It will in the future be useful in neighborhoods that are too distant for Internet access through cable or xDSL and where the cost of laying or upgrading landlines to broadband access would be expensive.

1.1.5 Wireless Sensor Networks

Wireless sensor networks (WSNs) [18, 19] consist of a large number of sensor nodes. Sensor nodes can be deployed in many different places to sense environmental changes and report data to other nodes in a flexible network architecture. Sensor nodes are small and function like small computers. They usually comprise a processing unit, sensors, a communication device and a power source supported by a normal battery.

Sensor nodes use an ad hoc wireless type network to interchange information, each sensor supporting a multi-hop routing algorithm. Sensor nodes have features that make them suitable for deployment in hostile environments or over large geographical areas. Thus, wireless sensor networks are now widely used in many fields, including environment and habitat monitoring, healthcare applications, home automation and traffic control. Some examples are discussed below.

- Environmental observation: Sensor networks can be used to monitor environmental change, for instance to detect water pollution in a lake. Sensor nodes can be randomly deployed in unknown and hostile areas and relay the exact origin of a pollutant to a centralized authority, so that it can take appropriate measures to limit the spread of pollution. Other applications can include air pollution, forest fire detection and rainfall observation.

- Building monitoring: Thermostats and temperature sensor nodes can be used in large buildings or factories to monitor weather variation and to control for air condition.

They may also monitor for vibration that may cause damage to the structure of a building. Moreover, light sensor nodes can be used to monitor lighting equipment.

- Healthcare: Sensors can be used in biomedical applications to improve the quality of the provided care. Sensors can be implanted in the human body to monitor medical problems and record patients' body condition over a 24-hour period.

1.1.6 Radio Frequency Identification

Radio frequency identification (RFID) [20] is an advanced automatic identification technology. It is used mainly for automated data collection. RFID uses radio frequency waves to transfer data between a tag and a reader. As the tag enters the radio frequency (RF) field, the RF signal turns the tag on. The tag then transmits the ID and data to the reader. RFID readers translate the radio frequency information into digital information that can be read by software on the host computer. The computer determines the required actions and informs the reader to transmit data back to the tag. RFID readers are available in many sizes and shapes including portable units. All readers have the same basic architecture: an antenna, a decoder, a data converter, a computer interface, and a power supply.

RFID offers read/write capability, so users can add data to the tags as they pass by a reader; enabling functions like time stamping. RFID can be integrated with bar-code technologies to optimize data capture and exchange. Benefits include reduced human error, reduced labour costs, increased accuracy and opportunities to simplify existing procedures.

RFIDs have been implemented in many different areas. For instance, they can be used to identify, track, sort and detect a variety of objects, including people, vehicles, garments, containers and pallets. They can also be used in applications such as proximity access control, time-and-attendance management, vehicle identification, laundry identification, asset tracking, inventory control and factory automation.

1.2 Motivation

A novel network protocol might be a good way to enhance current Internet access in wireless links. Internet functions depend on good cooperation in different layers. In the current network, transport layer protocol is used to provide stable end-to-end transmission and ignores end hosts' features. Adjustments of transport layer only affect two end hosts and maintain the status quo of all internal nodes. Thus, adjustment of transport layer protocol is a good choice to enhance the network performance in the heterogeneous network but keeps minimal effect of the current network. TCP is the most popular transport layer protocol and TCP congestion control mechanism is an key mechanism to deal with the loss events. Thus, a well designed congestion control mechanism is one of the main issues as regards enhancing TCP performance in the heterogeneous network.

In order to understand how TCP functions, network simulators are often used. Network simulator is useful to evaluate new network protocols or mechanisms. Network simulator can be used to understand protocols under various conditions. But, network simulators carry restrictions. Wireless features such as quick handover and dynamic data rates cannot be realistically demonstrated by current simulators. A network emulator is another option for researchers who need realistic wireless simulations. However, previous emulators were designed for the simulation of wired networks and inner networks [21, 22, 23, 24, 25]. Their frameworks have poor extensions for the simulation of application programs and transport layer protocols; giant PC clouds must also be maintained, so that a simulated topology of a complex network (multiple nodes and links) can be achieved. The integration of new functionality is complex; thus, further developments are restricted. Hence, an emulator facilitates future research of TCP in heterogeneous networks. It will be proposed in Chapter 3 and 4.

1.3 Objectives

The goal of this thesis is to improve TCP delivery performanceover heterogeneous networks. There are two main parts. Firstly, a new TCP/IP network simulator will be proposed. It can simulate complex TCP behaviours in heterogeneous networks by using only simple devices, easy set-up and small demonstrated space. Thus, the simulator can simplify future TCP/IP research.

Secondly, a new proposal of wireless TCP will be proposed. It should enhance TCP performance in the heterogeneous network. The proposal focuses on the adjustment of TCP congestion control mechanisms at the TCP sender side and keeps remnant TCP mechanisms changeless at access points (APs) and the TCP receiver side, so the mechanism is compatible with the current Internet and only adjusts network codes in slight ways.

1.4 Contributions

The contributions of the thesis are as follows:

- A novel prototype of TCP/IP network simulator has been designed in [26, 27].
- A TCP/IP network emulator is presented as in [28].
- TCP/IP challenges and possible solutions in heterogeneous networks are discussed as in [29].

- TCP NewZag is designed to improve TCP performance in the heterogeneous network [30].

1.5 Thesis Overview

The rest of this thesis is organized as follows. Chapter 2 is the literature review. The chapter discusses proposed solutions to improve TCP performance over heterogeneous networks. Advantages and drawbacks of solutions are also compared in this chapter. Some network simulators are also discussed and compared. Chapter 3 presents a new Linux TCP/IP network simulator (LTCP). Chapter 4 introduces a framework for a wireless TCP simulator, an extension of the simulator described in Chapter 3. Chapter 5 proposes a new end-to-end loss differentiation algorithm (LDA), named TCP NewZag, to improve TCP performance in the heterogeneous network. Finally, Chapter 6 summarizes the major contributions and discusses future research directions.

Chapter 2

Literature Review

Wireless communication technologies have developed rapidly and wireless Internet access has become common extremely in recent years. In the wired network, Transmission Control Protocol (TCP) governs most Internet traffic, and it is also straightforward to use it for wireless access. Current wireless Internet access usually coordinates with wired backbone networks, and applications use the client server model, where the server (file storage) is located in the wired network (fixed host) and the service file is requested from a mobile device. Thus, TCP must enable to serve wireless Internet access through a heterogeneous network, namely a mixed wired and wireless network environment. The role of TCP under the application layer is to provide stable and correct end-to-end data transmission; thus TCP interaction with other wireless protocols is the subject of widespread study to enhance data access speed in wireless communications.

TCP was initially designed as an end-to-end protocol for wired networks. TCP implementations include different mechanisms to enhance the stability of transmission and congestion control is one important aspect. Congestion control mechanisms perform well in wired networks and support TCP transmission stability when a congested network is detected. However, these mechanisms suffer from extra challenges in the heterogeneous network, because of the different features of wired and wireless networks. A high performance congestion control mechanism of TCP, suitable for the heterogeneous network, is a main goal as regards enhancing TCP performance and keeping up to speed with the increase in wireless Internet access.

Network simulators are useful tools for analyzing and testing various network environments. Simulators can reduce network hardware cost if a wide range network topology is considered. Many well-known TCP evaluations have been conducted using simulation. In order to understand how TCP functions, several well-known network simulators are discussed in this section. But, network simulators carry restrictions. Thus, the network emulator is another option. Network emulators are designed for real network environments. In contrast with simulators, emulators can be constructed in a processor or multi-processors. Several well-known network emulators are also discussed.

This chapter provides details of TCP congestion control mechanisms. It starts with a discussion of the interaction between flow control and congestion control. Next, it talks about the features of wireless links, and follows with a discussion of TCP retractions over the heterogeneous network. Next, research focusing on TCP restrictions over heterogeneous links is discussed. Afterwards, network simulators and emulators are discussed. Some concluding remarks encapsulate the issues of main concern for the subsequent stage of research.

2.1 Generic Architecture

The generic architecture throughout the thesis is introduced in Figure 2.1. TCP was initially designed as an end-to-end protocol for use only in the wired network. In Figure 2.1, for example, TCP is connected between fixed host 1 (FH1) and FH2 in the wired network. The wired network comprises FH1, FH2 and thousands of intermediate nodes (Rx) and links. Its topology makes observation of TCP behaviour and performance a challenging task in the real wired network. In Chapter 3, a TCP network simulator (LTCP) was designed and introduced. LTCP can simulated a complex wired network topology in a Linux processor. Thus, LTCP can easily support observation of TCP behaviours and performance.

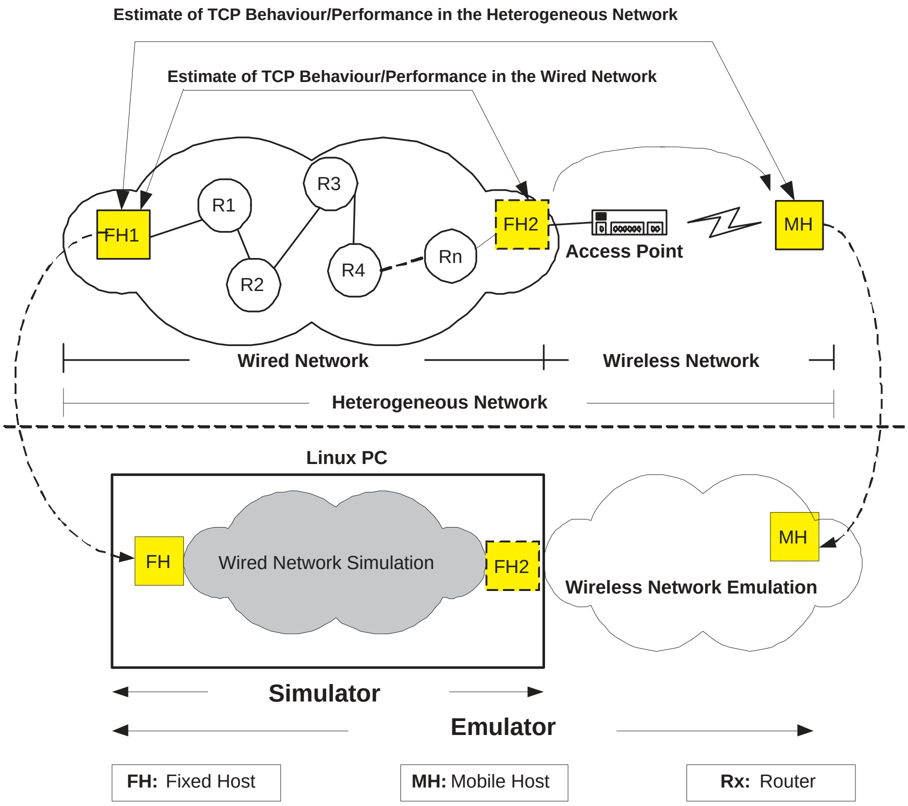


Figure 2.1: A generic architecture for the research throughout the thesis.

Wireless communication has grown rapidly in recent years. TCP connection has been extended to the wireless end and can connect between a fixed host and a mobile host (MH), in what is described as a heterogeneous network in the thesis. For instance, the transmission between FH1 and MH is a TCP connection in the heterogeneous network. The role of FH2 is taken over by MH, as shown in Figure 2.1. A access point is used to connect the wired network and the wireless network. In the heterogeneous network, the observation of TCP must include both the wired network and wireless network. Thus, in Chpater 4, a TCP network emulator, an extension of LTCP, was designed to connect with the real wireless network, so users can observe TCP behaviours and performance between FH1 and MH. A detailed description of simulator and emulator is given later.

Following the TCP studies in Chapter 2, Chapter 3 and Chapter 4, a new wireless TCP mechanism, named NewZag, was introduced in Chapter 5. NewZag can improve TCP performance in the heterogeneous network. The following sub-sections first introduce TCP, then TCP challenges are discussed in the heterogeneous network. Some wireless suggestions for TCP are introduced, and some famous network simulators are discussed and compared.

2.2 TCP Overview

TCP/IP was initially designed for data communication in the U.S. Department of Defence (DOD). In the late 1960s, the Advanced Research Projects Agency (ARPA) of the DOD entered into partnership with U.S. universities and the corporate research community to design open and standard protocols. The participants planned ARPANET, the first packet switching network, an experimental version of which went into operation in 1969. The experiment was a success, and the network became established.

In 1973, Vinton G. Cerf and Robert E. Kahn joined the ARPA project. They tried to develop an open architecture network model for heterogeneous networks to communicate with each other independent of individual hardware and software configuration, and with sufficient flexibility and end-to-end reliability to overcome transmission failures and disparities. Their collaboration led to the realization that a "gateway" was needed between each network to accommodate different interfaces and data packets routes. This meant designating host computers on a global Internet, for which they introduced the notion of an Internet Protocol (IP) address. In 1974, a new set of core protocols for ARPANET was proposed in a paper by Cerf and Kahn. The official name for the protocols was TCP/IP Internet Protocol Suite, commonly referred to as TCP/IP, which is taken from the names of the network layer protocol (Internet protocol [IP]) and one of the transport layer protocols (Transmission Control Protocol [TCP]) [31, 32].

The fundamental TCP specification document written by Jon Postel has been published in [33]. This document is an important part of the Internet protocol suite's core. It describes the TCP state machine, packet format, event processing, TCP's semantics for data transmission, flow control, multiplexing and acknowledgment. It is worth noting here that TCP congestion control mechanism is not described in [33], but, this document is still fundamental to modern TCP. [34] updates and clarifies [33], fixing some specification bugs and oversights. It also explains Jacobson's RTO estimation algorithms [35], as described in the next section.

2.2.1 TCP Control Mechanisms

Flow control and congestion control mechanisms are involved in TCP for maintaining stable and reliable network transmission. Flow control mechanisms try to prevent the possibility of the sender overflowing the receiver's buffer by restricting the rate at which TCP segments are sent. However, the transmission can also be throttled because of a crowded connection. In order to reduce the effect of congestion, TCP also carries a congestion control mechanism. The two mechanisms are described in the following two sub-sections.

TCP Flow Controls

The flow control mechanism limits the TCP sending rate. It maintains a receive window at the TCP sender site. The receive window is used to give the sender an idea about how much free buffer space is available at the receiver site. Several variables determine how TCP accomplishes the flow control mechanism, and these are defined below. Suppose host A is sending a file to host B over a TCP connection. We define the following terms [32]:

- *RcvBuffer*: the receive buffer size that host B allocates to the connection.
- *LastByteRead*: the number of the last byte in the data stream read from the buffer by the application process in B.
- *LastByteRecvd*: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B.

The receive window, denoted *RcvWindow*, is set to the amount of spare room in the buffer:

$$RcvWindow = RcvBuffer - [LastByteRecvd - LastByteRead] \tag{2.1}$$

RcvWindow is dynamic, due to the spare room changing with time. The variable *RcvWindow* is illustrated in Figure 2.2.

Host B informs host A how much spare room is available in the connection buffer, by placing the current *RcvWindow* in the receive window field of every segment sending to

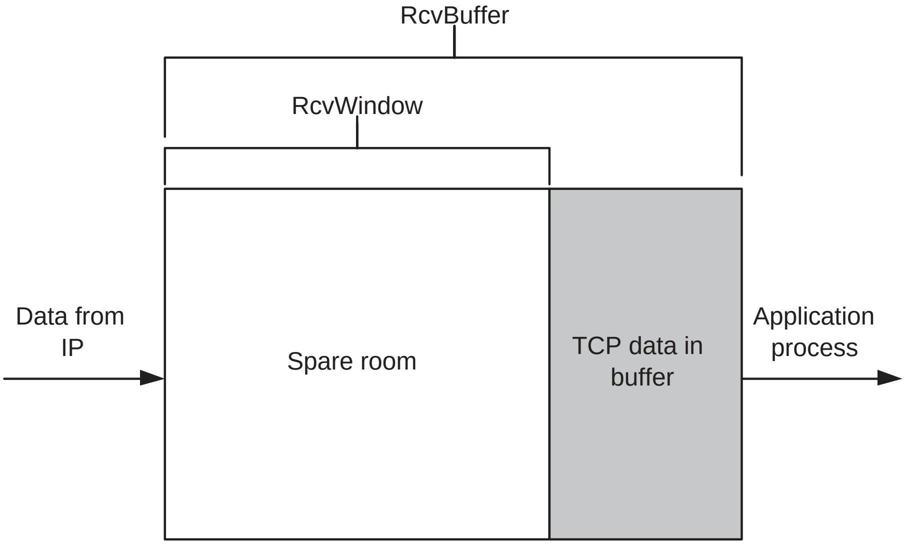


Figure 2.2: The receive window (*RcvWindow*) and the receive buffer (*RcvBuffer*).

A. Host A continues to track another two variables, *LastByteSent* and *LastByteAcked* (last byte acknowledged). The difference between *LastByteSent* - *LastByteAcked* is the amount of unacknowledged data that A has sent into the connection. By keeping the amount of unacknowledged data less than the value of *RcvWindow*, host A is sure that it is not overflowing host B's receive buffer. Thus, host A makes sure throughout the connection's life that:

$$LastByteSent - LastByteAcked \leq RcvWindow \tag{2.2}$$

Therefore, TCP can control the sending rate without overflowing the receiver's buffer.

TCP Congestion Controls

Although the congestion control mechanism is not described in [33], it is a required component of current TCP implementations. The congestion control mechanism is described in [36], which references Van Jacobson's congestion avoidance and control mechanisms for TCP [35]. A well-known TCP Reno has a number of behaviours described in [36]. The name “Reno” comes from the Net/2 release of the 4.3 BSD operating system. A detailed description of Reno is given later.

TCP involves the congestion control mechanism reducing the sending rate when a network congestion has been detected on the intermediate links. Section 2.2.1 states that a TCP connection consists of a receive buffer, a receive window, and several variables (*LastByteRead*, *LastByteAcked*) for control of traffic flow. The TCP congestion control mechanism keeps track of an additional variable, the congestion window (*cwnd*), which imposes a constraint on the rate at which a TCP sender can send traffic into the network. The amount of unacknowledged data that a sender may not exceed is the minimum of either *cwnd* or *RcvWindow*, that is :

$$LastByteSent - LastByteAcked \leq min\{cwnd, RcvWindow\} \tag{2.3}$$

If we focus on congestion control and suppose that the TCP receive buffer is so large that it can be ignored, Equation 2.3 can be displayed as [32]:

$$LastByteSent - LastByteAcked \leq cwnd \tag{2.4}$$

Equation 2.4 limits the amount of unacknowledged data at the sender; thus, it limits the sending rate. In recent years, different TCP congestion control mechanisms have been discussed. [37] suggests that congestion control uses the number of bytes acknowledged instead of the number of acknowledgments received. This suggestion has been implemented in Linux operating systems. [38] suggests a modification to TCP's steady-state behaviour to use very large windows efficiently.

Few popular and well-known TCP algorithms, TCP Tahoe [35], TCP Reno [36], TCP NewReno [39, 40], TCP Selective Acknowledgement [41, 42, 43] and TCP Vegas [44], are used as examples of TCP congestion mechanisms, as reviewed in the next sections.

2.3 TCP Tahoe and TCP Reno

TCP Tahoe [35], proposed by V. Jacobson, assumes that network congestion signals are represented by lost packets. Three main mechanisms are implemented in Tahoe: slow-start, congestion avoidance and fast retransmit; a number of modifications to Jacobson's initial algorithms are described in [45]. Tahoe also implements the round-trip time (RTT) variance estimation to reset the retransmission timeout (RTO) value. TCP Tahoe was first implemented in 4.3 BSD Tahoe TCP in 1988.

TCP Reno retained the enhancements incorporated into Tahoe, but modified the fast retransmit operation to include fast recovery [46, 47], which should be activated after a fast retransmit. There are some further differences between Tahoe and Reno in how they detect and react to packet loss, as will be discussed later. Reno is one of the most popular TCP algorithms, and has been widely deployed on the Internet. Even though many new TCP technologies have been proposed to enhance its performance, its congestion control and retransmission mechanisms are still the foundation stone for current TCP algorithms.

This section discusses the behaviour of Tahoe and Reno. The two TCP mechanisms operate similar in slow-start, congestion avoidance and fast retransmit algorithms, so we will discuss them together and point out the differences.

2.3.1 Slow-start and Congestion Avoidance Algorithms

The slow-start phase is triggered at TCP connection initiation or after retransmission timeout. The main objective of this phase is to probe the available network bandwidth. When a TCP connection begins, the TCP sender initializes the *cwnd* to one maximum segment size (MSS), and continues to increase its sending rate exponentially until a loss event is detected. During this phase, the TCP sender injects packets at a slow rate (one MSS), but then increases sending at an exponential rate. More specifically, when the first segment is acknowledged, the TCP sender increases the *cwnd* by one MSS and sends out two segments. If these two segments are acknowledged, the TCP sender increases *cwnd* by one MSS for each of the acknowledged segments; thus, the *cwnd* will become four MSS, so the value of *cwnd* doubles every round-trip time (RTT) during the slow-start phase. Three modifications of *cwnd* and slow-start phase are described in [48, 49, 50]. [48] suggests reducing the congestion window over time when no packets are flowing. This violates TCP behaviour defined in [36], which TCP sender should set its congestion window to the initial window after and idle period of an retransmission timeout. [49] describes a more conservative slow-start behaviour to prevent massive packet losses when a connection uses a very large window. [50] updates [36] to permit an initial TCP window of three or four segments during slow-start phase, depending on the segment size.

While the exponential growth of *cwnd* can help TCP reach an appropriate speed from a slow-start beginning, it also causes buffer overflow problems at the intermediate nodes. To overcome this, a sender-estimated slow-start threshold (*ssthresh*) is proposed to reduce the growth of *cwnd*. When the *cwnd* reaches *ssthresh*, TCP stops the exponential growth of *cwnd* but linearly increases the *cwnd*. More specifically, the *cwnd* grows more conservatively, by only 1 MSS every RTT. The initial *ssthresh* is set to an arbitrary value depending on the implementation of the operating system. This is the so-called **congestion avoidance phase**.

2.3.2 Round-trip Time Estimation

Round-trip time (RTT) estimation [34, 51] is used for determining the retransmission timeout interval. The sample RTT, denoted *SampleRTT*, for a segment is the amount of time from when the segment is sent until an ACK for the segment is received, and *EstimatedRTT* is the average value of *SampleRTT*. When TCP obtains a new *SampleRTT*, TCP updates *EstimatedRTT* according to the Equation 2.5. The recommended value of α is $\alpha = 0.125$ [51].

$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT \tag{2.5}$$

In addition, [51] defines $DevRTT$ as an estimate of how much $SampleRTT$ deviates from $EstimatedRTT$, presented in Equation 2.6.

$$DevRTT = (1 - \beta) \times DevRTT + \beta \times \mid SampleRTT - EstimatedRTT \mid \tag{2.6}$$

According to [51], the recommended value of β is 0.25. *DevRTT* is an exponential weighted moving average (EWMA) of the difference between *EstimatedRTT* and *SampleRTT*. If the *SampleRTT* values have little fluctuation, then *SampleRTT* will be small. On the other hand, if there is a lot of fluctuation, *SampleRTT* will be large.

2.3.3 Retransmission Timeout

A packet can be declared lost if the sender does not receive the corresponding acknowledgement (ACK) when the TCP timer expires. When a loss event occurs, the TCP sender must retransmit the lost packet. In Tahoe, a packet loss can only be recovered in two ways: by retransmission timeout (RTO) or with the fast retransmit algorithm. Tahoe [35] uses *EstimatedRTT* and *DevRTT* to estimate the interval of RTO, as defined in Equation 2.7.

$$RTO = EstimatedRTT + 4 \times DevRTT \tag{2.7}$$

If the sender enters the RTO phase, all lost packets must first be retransmitted. When all lost packets are completely retransmitted, TCP leaves the RTO phase and begins again from the slow-start phase. The RTO event is a signal that the current network is seriously congested. Thus, the sender slows down the transmission speed to react this phenomenon. However, the RTO event has an intense effect on the TCP sending rate, so the fast retransmit algorithm is implemented to efficiently retransmit the lost packet before the timeout event happens.

2.3.4 Fast Retransmit

The fast retransmit algorithm is another algorithm to deal with packet loss events. The fast retransmit algorithm can quickly respond to a lost packet before the RTO has expired. When an out-of-order packet is received, the receiver transmits a duplicate ACK. For instance, there are 4 packets (1 to 4) transmitted from the sender to the receiver, and a loss occurs with packet 2. On receipt of packet 1, the receiver should have sent out ACK 2, which means I received packet 1. When the receiver still does not receive packet 2 but packet 3 arrives, it will send out another ACK number 2 to notify the sender that the receiver is still expecting packet 2. To the sender, this is a duplicate ACK, because the sender has previously received ACK 2 after the receiver received packet 1. If packet 4 arrives while packet 2 is still missing, the receiver will send out another ACK 2. When the sender receives three duplicate ACKs, the correspondent packet (packet 2) will be marked as lost and the sender will retransmit the packet again, if the RTO has not expired. The behaviour of Tahoe and Reno differ in how they react to packet loss:

- Tahoe: Tahoe unconditionally cuts its *cwnd* to 1 MSS and enter the slow-start phase after either type of loss event (RTO and three duplicate ACKs).
- Reno: When the RTO event occurs, Reno operates the same as Tahoe. Unlike the RTO event, if three duplicate ACKs are received, Reno will halve *cwnd*, perform a fast retransmit, and enter a phase called fast recovery (Reno only).

2.3.5 Fast Recovery

Reno activates a fast recovery [47] phase after fast retransmit is triggered. Unlike Tahoe, Reno cancels the slow-start phase after a triple duplicate ACKs. The main reason for canceling slow-start here is that, even thought a packet has been lost, the arrival of triple ACKs indicates that some segments have been received at the sender. Thus, unlike in the RTO event, the transmission links are still capable of delivering some segments, even if some segment loss events have occurred. The canceling of the slow-start phase after a triple ACK is called **fast recovery**. In Reno, the fast retransmit and fast recovery algorithms are usually implemented together as follows [36].

1. When the third duplicate ACK is received, set $ssthresh = max(FlightSize/2, 2 * MSS)$. *FlightSiez* is the amount of data that has been sent but not yet acknowledged.
2. Retransmit the lost segment and set $cwnd = ssthresh + 3 * MSS$. This “inflates” the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.
3. For each additional duplicate ACK received, set $cwnd = cwnd + MSS$. This inflates the congestion window in order to reflect the additional segment that has left the network.
4. Transmit a segment, if allowed by the new value of *cwnd* and the receiver’s advertised window.
5. When the next ACK arrives that acknowledges new data, set *cwnd* to *ssthresh* (the value set in step 1). This is termed “deflating” the window.

2.3.6 TCP NewReno

NewReno [39, 40] is a Reno variant. The fast retransmit and fast recovery algorithms are known to generally not recover very efficiently from multiple packet losses in a single flight of packets [52]. Thus, TCP NewReno [39] proposed some modifications to address this problem. In NewReno, the Reno's "fast recovery algorithm" is modified to enhance TCP performance when multiple packet losses occur in a single congestion window. The difference between NewReno's and Reno's fast recovery algorithm is their corresponding reaction to a "new coming ACK". In NewReno, "new coming ACKs" are classified into two categories: "partial ACK", which is an ACK that represents only part of the outstanding segments and "new ACK", which means all data transmitted at the start of the fast retransmit phase have to be acknowledged. In Reno, when the sender receives partial ACKs, it leaves the fast recovery algorithm. According to [39], this process could trigger multiple TCP timeouts, because of multiple packet losses in a single *cwnd*. Thus, in NewReno, when a sender receives partial ACKs, it stays in the fast recovery phase until all outstanding segments have been retransmitted. NewReno only exits the fast recovery phase when it receives "the new ACK". The fast recovery modification of NewReno has been described in [40]. A TCP sender can use partial ACKs to make inferences determining the next segment to send in situations where SACK would help but is not available. Even though a slight modification is made in [40], the NewReno can make good performance when multiple segments are lost from a single window.

NewReno improves the throughput efficiency of TCP. Most of the modern operating systems, such as Windows and Linux, have used NewReno as their default TCP congestion control mechanism. Thus, NewReno will feature in future discussion in this thesis.

2.3.7 TCP Selective Acknowledgement

Selective Acknowledgment (SACK) [41] is also designed to solve the problem of multiple loss events. Compared with NewReno, SACK only retransmits segments that have actually been lost. NewReno implements a "cumulative acknowledgment scheme" in which received segments are not acknowledged if they are not at the left edge of the receive window. SACK implements an additional "SACK option" in the "Options field" of the TCP header. It invokes the most recently received packets and the most recently reported information on SACK packets. With the help of additional information, the TCP sender can retransmit "select packets" only, instead of "serial packets". Two main SACK drawbacks are discussed: header overheads and the cooperation of sender and receiver. The former describes the problem that if the connection state is stable, the significant overhead of TCP header (additional SACK field) decreases the usage of network bandwidth. In [53], header compression technology has been proposed to decrease the overhead. The latter problem is that a SACK connection can be set up only if the sender and the receiver are SACK supported. Two modifications of initial SACK are described in [42, 43]. [42] extends initial SACK to cover the case of acknowledging duplicate segments. [43] describes a relatively sophisticated algorithm that a TCP sender can use for loss recovery when SACK reports more than one segment lost from a single flight of data.

2.3.8 Summary

NewReno and SACK are both classified as end-to-end proposals, further discussed in Section 2.6.3 below. They were initially designed to improve TCP performance over the wired network and have been recently discussed along with TCP improvements in the heterogeneous network [54, 55, 56]. The advantage of these protocols is non-sensitivity with mobile nodes or mobile actions (e.g. handoff). If these protocols could perform well over the heterogeneous network, it is expected that only a few modifications would be necessary to extend them into wireless environments. Table 2.1 summaries different TCP algorithms discussed in this section.

Table 2.1: Comparisons between Tahoe, Reno, NewReno and SACK.

TCP version	Difference
Tahoe	Slow-start, congestion avoidance and fast retransmit
Reno	Base on Tahoe, added fast recovery
NewReno	Base on Reno, modified Reno's "fast recovery"
SACK	Base on Reno, added SACK function (select acknowledge)

2.4 TCP Vegas

TCP Vegas [44] involves a different congestion control mechanism. It primarily modifies congestion avoidance and the fast retransmit algorithms of TCP Reno. In contrast with Reno, Vegas can increase or decrease *cwnd* by one MSS every RTT, depending on a probe of "backlog value", which is the amount of packets queued at intermediate nodes. When the backlog value is larger than a defined threshold, Vegas decreases the *cwnd*; when the backlog value is smaller than the threshold, it increases the *cwnd*. Because Vegas' congestion control algorithm can efficiently prevent packet loss, Reno's retransmission algorithms are infrequently invoked, so the average throughput of TCP might increase in some situations. The idea of Vegas' congestion control mechanism has been widely discussed in recent TCP implementations.

2.4.1 The Basic Principle of Vegas

Vegas uses a measure of the variation in RTT to indicate congestion in the network. Current wired links are built upon modern physical media, which support very stable data transmission. Thus, the probability of bit corruption or transmission loss are extremely small, because of medium error. Packet loss always occurs at an intermediate node, due to buffer overflow. The relationship between throughput, link load and queuing state, according to [57] is shown in Figure 2.3 (a); and the relationship between RTT, link load and queuing state is shown in Figure 2.3 (b). When the amount of link load is small, the throughput increases synchronously with the link load, as shown at label 1 in Figure 2.3 (a). The variation in RTT (Figure 2.3 (b)) is also small at this time, as shown at label 1. However, when the incoming rate is higher than the network capacity, packets must be queued inside the router buffer, and the network becomes congested. This is the *initial point* of congestion. When the network is congested, the subsequent packets experience queuing delay, which results in an increase in RTT, as shown in Figure 2.3 (b), label 2. Thus, the measured variation of RTT around the *initial point* can be used as a signal to indicate that the state of the current network is congested, so the *cwnd* can be strategically adjusted to increase or decrease the transmission speed. If the number of queuing packet exceeds a queue limited threshold, packets may be dropped. According to different queuing disciplines, the dropped packets may be randomly chosen or be the last packet of the queue [58]. When packets are dropped from the queue, throughput decreases quickly due to packet retransmission, and RTT increases because of long queuing delay, as shown in Figure 2.3, label 3.

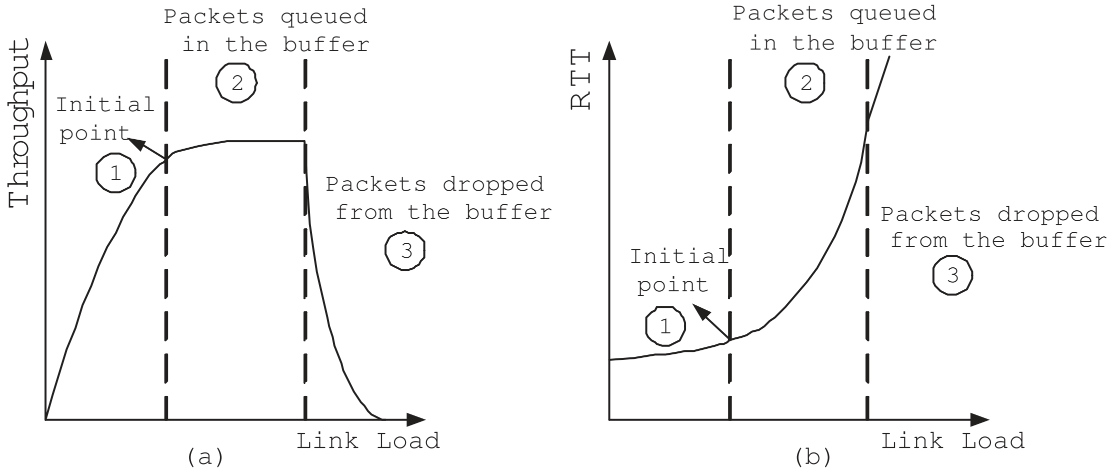


Figure 2.3: (a) The relationship between throughput and link load. (b) The relationship between RTT and link load.

2.4.2 Vegas Congestion Avoidance Mechanism

TCP Vegas does not continually increase the *cwnd* during the congestion avoidance scheme. Instead, it detects incipient congestion by comparing the *actual throughput* or *measured throughput (ActTput)* to its notion of *expected throughput (ExpTput)*. In Vegas, the sender measures the so-called *ExpTput* and *ActTput* rates according to Equation 2.8.

$$\begin{aligned} ExpTput &= cwnd/BaseRTT \\ ActTput &= cwnd/RTT \end{aligned} \tag{2.8}$$

The *BaseRTT* is the minimum measured round-trip time. It is the RTT of the first segment sent before packets are queued in the router buffer due to network congestion. The *RTT* is the smoothed round-trip time. Vegas compares *ActTput* to *ExpTput*, and adjusts its *cwnd*. It states that *Diff* = *ExpTput* - *ActTput* and α, β are two thresholds. When *Diff* $< \alpha$, Vegas increases the *cwnd* linearly during the next RTT; when *Diff* $> \beta$, Vegas decreases the *cwnd*. Vegas leaves the congestion window unchanged when $\alpha <$ *Diff* $< \beta$. Vegas tries to avoid congestion by evaluating the available bandwidth of the link. Details of Vegas' schemes are described follows.

2.4.3 Vegas Slow-start Mechanism

The Vegas implementation of the slow-start mechanism is similar to that of Reno. During the slow-start phase, Vegas doubles the *cwnd* every other RTT, as opposed to Reno's every RTT. But, as soon as Vegas detects queue build up during the slow-start phases, it changes the state immediately from the slow-start to the congestion avoidance phase.

2.4.4 Vegas Retransmission Mechanism

TCP Reno, NewReno and SACK use loss of packets as a signal to detect the appearance of congestion in the network; on the other hand, Vegas tries to prevent network congestion to reduce packet loss. Reno uses two mechanisms, RTO and fast retransmit, to detect and retransmit the lost packets. Vegas extends Reno's retransmission mechanisms with only slight adjustments. Vegas proposes a *timestamp value*, which records the sending time of each packet, to determine more accurate RTT estimates and retransmit the packet in two situations:

- When a duplicate ACK is received, Vegas checks the difference (*D_Time*) between current time and the timestamp recorded for the corresponding packet. If *D_Time* is greater than the timeout value, Vegas retransmits the packet without waiting for three duplicate ACKs.
- When a non-duplicate ACK is received, if it is the first or second one after retransmission, Vegas again checks to see whether the time interval since the segment was sent is larger than time timeout value. If it is, Vegas retransmits the packet. The objective of this phase is to retransmit the lost packet without waiting for the duplicate ACK.

The Vegas' retransmission mechanisms not only reduce the time to detect lost packets but also to detect lost packets even though there may be no second or third duplicate ACK.

2.4.5 Summary

TCP Vegas proposes a new congestion avoidance mechanism to prevent packet loss and a new retransmission mechanism to speed up the retransmission time. Thus, TCP Vegas can perform well in some wired links. According to [24, 44], by reducing packet loss and subsequent retransmissions, Vegas improves throughput in a range from 37% to 71% when a queue overload occurs in a link, compared to Reno. However, if the link is smooth, without serious queue overloading, Vegas' congestion control mechanism could reduce throughput due to conservative estimates of link bandwidth. This phenomenon will be discussed in Section 5.4. Besides, Vegas is not applicable to networks with asymmetric routing, which means that data packets and ACKs are routed from different paths. In this situation, the Vegas queuing algorithm will fall into wrong evaluation of current path state. Moreover, Vegas can only prevent packet loss due to buffer overflow at intermediate nodes. When the transmission involves the wireless network, packet loss can be caused by other factors. Hence, it is difficult for the current Vegas algorithm to prevent packet loss in advance. Thus, a degradation in TCP performance may be expected.

2.5 TCP Involvement in Wireless Communication

2.5.1 Features of Wireless Communication

Before discussing TCP connection to wireless links, some wireless features that differ from wired links are first introduced. We focus on those that could cause the packet retransmission.

- User mobility: The user mobility between wireless cells could cause handoff or handover of transmission state. Handoff is to transfer the controller status between different base stations (BS). The infrastructure of wireless Internet is used to connect mobile terminals to the Internet. The base station, which provides radio signals to the mobile host (MH), is connected with a special wireless gateway. The wireless gateway is used for communication between wireless networks and backbone interconnection. When an MH travels between two positions, its service can be supported by different BSs, and connection status is transferred at the same time. This is explained as mobile handoff or handover.
- Temporary disconnections: According to [59], temporary disconnections are in two basic types:
  1. System managed disconnections:
     - Non-lossy handover: The MH can receive all packets from the sender in this case. There is only packet delay, and no packet loss, when the handover

    occurs between an intermediate host and an MH.

    - Lossy handover: The MH cannot receive all the packets from the sender. Some packets get lost due to the handover. The lost packets need to be retransmitted again, and end-to-end throughput slows down.

  2. Disconnections due to link error: In this case, disconnections do not come from the handover but occur due to link errors. Hence, packets are lost and are retransmitted by the sender.

- High bit error rates: Wireless transmission is easily affected by other signals so that bit corruptions are high.

- Temporary bandwidth overloading: A wireless local area network could suddenly be connected by a large number of mobile users at the same time. This situation could cause temporary network congestion and mass packet loss events.

- Dynamic network route: The movement of an MH requires a dynamic network routing path, because the MH changes position from one wireless domain to another. In this case, the new routing path must be re-calculated for the following transmissions, so additional time is necessary for packets to stay in the buffer of intermediate nodes and wait for a new routing path. Since limited buffer size is supported by the intermediate nodes, a buffer overflow could occur and cause packet dropping.

- Bandwidth Delay Product: In some special wireless links, such as 2.5G/3G networks or satellite networks, the link bandwidth delay product (BDP) tends to large [54, 60]. The BDP determines the amount of data that can be in transit in the network. It is the product of the available bandwidth and the latency, or RTT. BDP is a very important concept in a window based protocol such as TCP. But, the basic implementation of TCP is unsuitable for high BDP networks [33], and therefore some modifications should be made to enhance the TCP performance [61]. However, large BDP could also appear in wired networks, so we will not discuss the BDP feature in the thesis.

Except the BDP, all of the above factors could lead to the packet loss, and packet retransmission becomes necessary. The factor of temporary bandwidth overloading is similar to wired network congestion, while the other factor are distinctive wireless features. As defined in [62], such packet loss will be referred to as random packet loss in this thesis.

2.5.2 The Wireless Effect on TCP Congestion Control Mechanisms

In the wired network, the main reason for packet loss is a congested network. Thus, it is appropriate that packet loss is used as a feedback message to slow down the sending rate, so that the congested situation in the middle network can be eased. When TCP is extended to the wireless end (as discussed in Section 2.5.1), additional packet loss might occur, due to wireless features. Even though each loss event is caused by different errors and factors, they still affect the TCP congestion control mechanism. Thus, the TCP congestion control mechanism resets the *cwnd* using the slow-start algorithm, which comes from the idea of congestion avoidance, as discussed in Section 2.2.1. This results in an unnecessary reduction in the link's bandwidth utilization, thereby causing a significant degradation in throughput and very high interactive delays [63]. Therefore, the congestion control mechanism becomes inappropriate when TCP is running in a heterogeneous network environment.

2.6 Suggested Solutions for Wireless TCP

Some solutions have been proposed to improve the performance of TCP in the heterogeneous network. According to [59, 64], these approaches can be classified into three basic groups: link layer proposals, split-connection and end-to-end proposals. Figure 2.4 shows some aspects from selected studies on these proposals. Because the present study focuses on the congestion control mechanism, the link layer proposals are only briefly introduced, the split-connection proposals are used as a contrast, leaving the way clear to end-to-end proposals.

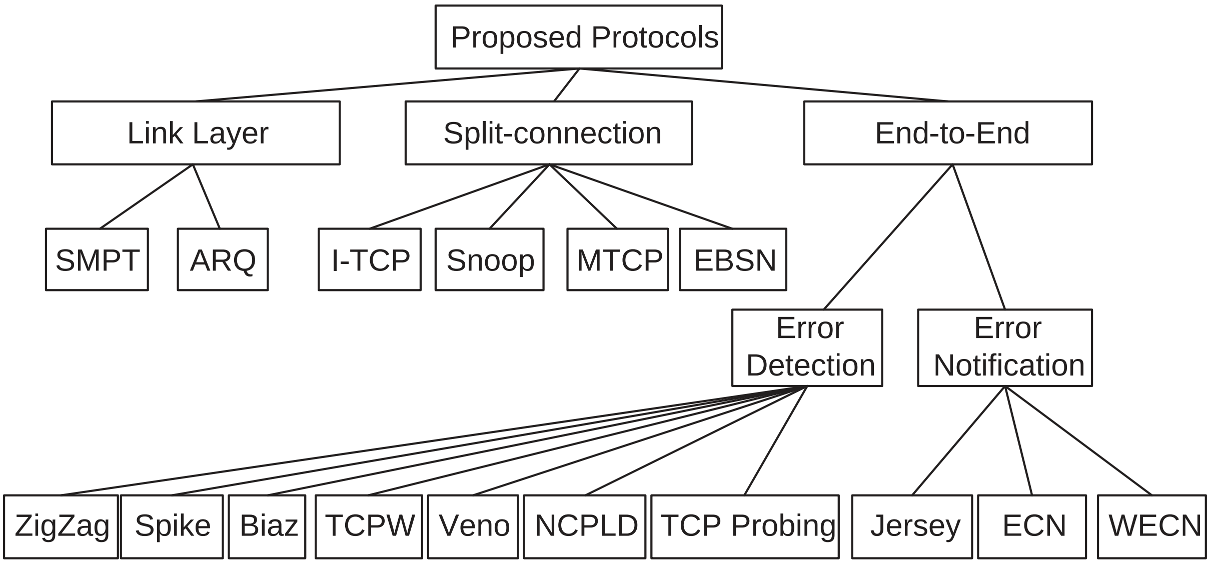


Figure 2.4: Protocols to improve the performance of TCP over the heterogeneous network.

2.6.1 Link Layer Proposals

The link layer proposal tries to solve problems at the link layer. It hides error events occurring in wireless links from upper layers such as TCP. Most of the proposals focus their solutions at the local link layer of wireless networks. In general wireless systems, the lower part of the link layer is divided into Radio Link Control (RLC) and Medium Access Control (MAC). Many link layer proposals are based on the cooperation of these two layers with TCP to improve TCP performance in the heterogeneous network. Automatic repeat request (ARQ) [65] and Simultaneous MAC Packet Transmission [66] are two aspects of research worth noting in the link layer domain.

There are two advantages improving TCP performance based on a link layer approach. First, TCP does not need to change. This is important because TCP is a widely implemented transport protocol over the Internet. Secondly, new approaches can be implemented only at the local link layer of the wireless links in which they are still under development, and have not been used commonly as TCP. The main drawback of link layer proposals is that link layer retransmissions could cause extensive delay variation at the IP level. Some possible solutions have been presented in [66, 67].

2.6.2 Split-connection Proposals

Split-connection proposals tries to hide the wireless part from the wired network by separating the flow control at the intermediate router (or a base station), so that the wireless behaviour has the minimum impact on the wired network. Under split-connection proposals, a TCP connection is separated into two different phases. From the fixed host (FH) to intermediate node (i.e. base station) is the first portion, which implements regular TCP without any difference. From intermediate node to the MH is the second portion, which constructs a new or modified TCP to enhance TCP performance in the wireless links. The split-connection proposal is illustrated in Figure 2.5.

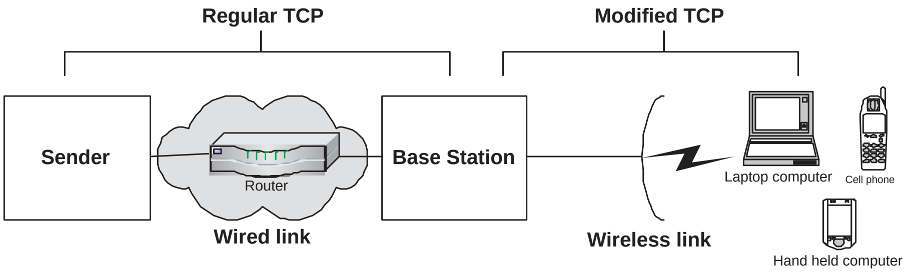


Figure 2.5: The split-connection mechanism implemented in the heterogenous network.

Split-connection proposals can be classified into those that violate end-to-end semantics, and others that maintain it [59]. The following split-connection protocols are described below: Indirect-TCP, Snoop Protocol, MTCP and Explicit Bad State Notification (EBSN).

- Indirect-TCP [68]: The Indirect-TCP (I-TCP) mechanism was the first proposal to implement the split-connection method. It allows two independent transport layer protocols between mobile support router (MSR) or BS, and the MH and between the FH and the MSR. In I-TCP, the MSR connects the FH to the MH, and establishes two separate TCP connections with the FH and MH, respectively. The MSR communicates with the FH on behalf of the MH. The TCP congestion window is maintained separately for the wired and wireless networks. Thus, if the MH changes to another cell, a new MSR could take over the communication with the FH seamlessly. Hence, the FH is hidden from the unreliable feature of wireless networks. Compared with regular TCP, I-TCP has performed 1.5 to 4 times better performance in end-to-end throughput [68]. The implementation of I-TCP is also presented in [69]. The drawback of I-TCP is losing the end-to-end semantic and will later discuss in this section.

- Snoop Protocol [70]: The Snoop protocol modifies the network-layer software and introduces a new module, call the snoop agent, at BS. The agent is designed to cache and store packets that are sent from the FH. When ACKs are sent from the MH, the agent interprets and compares the sequence number of each ACK with its stored packets. In this way, the snoop agent can detect duplicated ACK events or timeout events quickly and retransmit packets from its buffer instead of FH's. It acts as a middle TCP fixed host in the intermediate link and responds to packet loss events more quickly. Decreasing the retransmission time at BS not only reduces end-to-end delay but also increases bandwidth usage on wireless channels. This improves TCP end-to-end performance without losing the original architecture, and makes a minimum modification to the existing TCP implementations by only changing code at the BS.

- Explicit Bad State Notification [71]: Explicit Bad State Notification (EBSN) extends the Explicit Congestion Notification (ECN) [72] protocol to reduce the injection of redundant packets from the FH when a wireless link is in a bad state (i.e. high bit error rate). In wired networks, ECN is used to give the sender explicit feedback information from routers, which helps the sender to eliminate the possibility of timeout events and dynamic resetting of the TCP timeout clock. It decreases unnecessary packet retransmission. Section 2.6.1 above discussed different implementation of link layer proposals to improve TCP performance in the heterogeneous network. However, those protocols could not solve the problem of TCP timeout events, which happen at the FH and cause redundant packet retransmissions. This wastes network bandwidth and is a significant problem, because the bandwidth of wireless networks is limited and expensive. EBSN has been implemented in the heterogeneous network, where the timeout would be reset at the FH during local recovery. It dynamically resets the TCP timeout event based on the feedback of EBSN information before the wireless link enters a bad condition. The TCP sender would not retransmit new packets and maintain the same congestion window. In regular TCP, EBSN has been proved to increase performance improvement up to approximately 100% in wide-area networks and up to 50% in local area networks. EBSN also proposes the effect of packet size variation in wireless networks. The results show that the optimal packet size over wireless links is dependent on the error situations in the wireless networks. If an optimal packet size is chosen for use in wireless networks, up to 30% transmission improvement can be gained. It decides an optical packet size by implementing a fixed table at the base station. The fixed table records the different characteristics of error conditions over wireless networks. When packets are going through the BS, the present wireless condition is mapped with the fixed table, and an optimal packet size is selected.

- MTCP [73]: MTCP introduces a new session layer protocol called Mobile Host Protocol (MHP) over TCP between the BS and the MH. It proposes two alternatives for improving TCP performance over the MH. The first alternative is "MTCP", which establishes two different connections on the session layer protocol. One connection is from an MH to BS and the other connection is from the BS to the FH. An agent is implemented at the BS to act as a relay for traffic from the first connection to the other. If handoff happens, MHP can send an indication of "handoff in progress" to higher layers [73]. When the handoff completes, MHP transfers the connection state information to the new BS and establishes a new connection between the MH and its new BS. Therefore, no adjustment needs to be made at the FH, because MHP has taken over all new packet routing from the old BS to the new BS. The second alternative is named selective repeat protocol (SRP), which is similar to the first alternative, except that MHP uses a specialized protocol instead of TCP over the mobile host. SRP uses its own flow and error control mechanisms to prevent some unstable features in the wireless links. Both MTCP and SRP focus on the special behaviour of wireless networks, such as small maximum transmission unit (MTU), handoff and high error rates, and design new mechanisms to skip them. MTCP limits the performance degradation of TCP only in a "short" connection over wireless links. TCP traffic over "long" connection of wired links can be protected from the impact of unstable features in wireless links. Compared to regular TCP, MHP increases the end-to-end TCP throughput by avoiding unnecessary degradation of the congestion window. The advantage of MTCP is to improve TCP performance without modifying any existing flow and congestion control mechanisms.

The advantage of the split-connection proposal is to separate the TCP into two parts: the wired network TCP and the wireless network TCP. The wireless network TCP can be designed completely on the basis of on the features of wireless links. But, two drawbacks are often discussed:

1. Violation of the end-to-end semantic: A fundamental design philosophy of the Internet TCP principle is that it must maintain an end-to-end connection. In fact, it is this

principle that guarantees TCP delivery of data over any kind of heterogeneous network. Thus, the end-to-end principle [74] is very important. Under the end-to-end principle, the network is considered as a “black box”, which means that the TCP hosts cannot receive any explicit congestion information from the intermediate nodes. TCP probes the available sending rate at the sender site by increasing the input load until implicit feedback acknowledgements: the timeout event or duplicate ACKs. When the TCP connection is set up, packets from the sender first arrive at the BS. In split-connection proposals, three operations are performed at BS: creating new ACKs and sending them back to the sender, reproducing packets and storing them in the buffer, and sending packets to the mobile receiver. When the wireless network is in bad condition, ACKs may arrive at the sender before their correspondent packets actually arrive at the receiver. Hence, the end-to-end semantic is broken.

2. Optimum buffer space at the intermediate node: The buffer space at the intermediate node directly affects the success or failure of the split-connection proposal. As previously discussed, the intermediate node must reproduce packets and store them in the buffer to speed up retransmission when the loss events occur at the wireless part. Hence, a well-designed buffer space is important. If the buffer space is too small, packet loss occurs at the intermediate node because of buffer overflow. If the buffer size is too large, it could increase the average waiting time of packets in the buffer. Thus, the optimum buffer space is usually discussed in terms of the split-connection proposal.

2.6.3 End-to-End Proposals

End-to-end proposals attempt to enhance TCP performance at the TCP end hosts. Two mechanisms are usually implemented in the end-to-end proposals: loss discrimination algorithms (LDA) and the adjustment of current TCP congestion control mechanisms. Different mechanisms of LDA is used to distinguish congestion loss from random packet loss. LDA then reports the loss type to the TCP hosts, so the TCP can adjust the congestion mechanism according to the feedback information from the LDA. According to the implementation of LDA, two types of end-to-end approach can be classified: error detection approaches and error notification approaches.

1. Error detection approaches: TCP sender estimates the error type of the current network through feedback information from the receiver. In this approach, the modification is implemented only on the sender side, so the effect of current Internet status is minimum. TCP Westwood, Biaz , mBiaz, Spike, ZigZag, TCP Veno, NCPLD and TCP Probing are classified as this approach. Because TCP NewZag proposed in Chapter 5 refers Biaz, Spike and ZigZag, we only brief introduce four schemes here and will

later detailed described in the Chapter 5.

- TCP Westwood [75, 76, 77, 78]: TCP Westwood (TCPW) uses rate estimation methods to set the congestion window and slow-start threshold after a packet loss. The LDA of TCPW keeps the sender continuously monitoring ACKs from the receiver and computing its current Eligible Rate Estimate (ERE) [79]. ERE relies on an adaptive estimation technique applied to ACK stream. The goal of ERE is to estimate the connection eligible sending rate with the goal connection. In TCPW, the sender adaptively computes T_k, an interval over which the ERE sample is calculated. An ERE sample is computed by the amount of data in bytes that were successfully delivered in T_k. T_k depends on the congestion level, the latter measured by the difference between "expected rate" and "achieved rate" as in TCP Vegas. That is, T_k depends on the network congestion level as follows:

$$T_k = RTT \times \frac{cwnd/RTT_{min} - RE}{cwnd/RTT_{min}}, \tag{2.9}$$

  where $cwnd/RTT_{min}$ is the minimum RTT value of all acknowledged packets in a connection, and RTT is the smoothed RTT measurement. The expected rate of the connection when there is no congestion is given by $cwnd/RTT_{min}$, while RE is the achieved rate computed based on the amount of data acknowledged during the latest RTT, and exponentially averaged over time using a low-pass filter. When there is no congestion, and therefore no queuing time, $cwnd/RTT_{min}$ is almost the same as RE, producing small T_k. In this case, ERE becomes close to a packet pair measurement. On the other hand, under congestion conditions, RE will be much smaller than $cwnd/RTT_{min}$, due to longer queuing delays. As a result, T_k will computing the ERE closer to a packet train measurement. After computing the ERE samples, a discrete version of a continuous first order low-pass filter using the Tustin [80] approximation is applied to obtain smoothed ERE. However, most of the evaluations are based on the wireless link being the last link to the receiver.

- Biaz Scheme[81]: The Biaz scheme proposes a loss differentiation algorithm (LDA) based on packet inter-arrival time to distinguish congestion losses from random packet losses. According to the simulation results, the Biaz scheme can work effective when (i) the TCP receiver is located in the wireless end, (ii) the bandwidth of the wireless link is smaller than the bandwidth of the wired link, and (iii) the overall packet loss rate is small. But, the Biaz scheme might over-estimate numbers of wireless packet loss in some network topology. Thus, mBiaz[82] adjusts Biaz's LDA to enhance the accuracy of Biaz.

- Spike Scheme [83]: Spike is a rate control algorithm used for UDP flows. It uses spikes in relative one way trip time (ROTT) as a congestion signal. Tobe et al. find that sequences of the spike-trains (spikes) are only related to congestion packet losses and are not related to random packet losses. Thus, the spike-trains are used to classify link paths, allowing for the use of different congestion control mechanisms on different paths.

- The ZigZag Scheme [82]: ZigZag uses a similar notation as in the Biaz scheme. In ZigZag, losses are classified based on the mean and deviation of ROTT for different number of lost packets. According to [82], the misclassification rate of ZigZag is rather insensitive to changes in network topology. Besides, it does not work well under some network situations [84].
- TCP Veno [85]: TCP Veno uses the estimate of the *backlog* value as a signal to differentiate congested losses from random packet losses. The *backlog* value was first developed by TCP Vegas, as discussed in Section 2.4. If the *backlog* value is below the threshold, the loss is considered as random packet error. Otherwise, the loss is regarded as congested error. If the loss is considered congested, Veno maintains its congestion control mechanism as standard Reno. If the loss is due to a random error, it increases the *cwnd* following the new strategy.
- Non-Congestion Packet Loss Detection [86]: Non-Congestion Packet Loss Detection (NCPLD) implicitly detects the type of packet loss using the variation of delay experienced by TCP packets. The implementation uses a conservative approach, which considers that congestion avoidance must be applied, if any congestion is detected in the network. However, it does not perform well when the network is congested. NCPLD evaluates a “delay threshold” from the *knee point* [87], which is similar to the *initial point* described in Section 2.4.1. On detection of a packet loss, the sender compares the currently measured round trip delay with the delay threshold. If the measured value is less than threshold, it means that the bandwidth is still available on the current network. Hence, NCPLD assumes that the packet loss is a non-congestion loss and retransmits lost packets using the fast retransmission mechanism. Thus, the *cwnd* is maintained. Otherwise, it assumes that the network is congested and the *cwnd* is reduced.
- TCP Probing [88]: In TCP Probing, a “probe” mechanism is implemented to replace the normal fast retransmit and fast recovery mechanism. When three duplicate ACKs are received, the sender runs into a state named “probe cycle” and dispatches a probe packet instead of retransmitting the lost packet. The probe packet is used to test the current RTT. If the first probe packet is lost, a series of probe cycles will be activated. Once these probe actions have been completed, the current network situation can be noted via the measured RTTs. If the reason for the packet loss is network congestion, the *cwnd* is decreased. Otherwise, TCP maintains the size of the current *cwnd*.

2. Error notification approach: In error notification proposals, some special labels are implemented at the IP or TCP header for additional error messages.

   - TCP Jersey [89]: Two LDA schemes are developed in TCP Jersey: congestion warning (CW) and available bandwidth estimation (ABE). CW is a packet marking scheme, which marks all packets when the average queue length exceeds a threshold. The purpose of CW is to inform the sender using a simple image of the bottleneck queue, so the sender can adjust *cwnd* according to the feedback information from the CW scheme. Jersey also develops an ABE module, which is comparable, and functions similarly to TCPW's ERE, but with different implementation. Jersey adopts NewReno's slow-start and congestion avoidance scheme based on its CW and ABE. If an ACK is received without the CW mark, it proceeds with TCP behaviour as NewReno. If the ACK or the third duplicate ACK is marked with the CW bit, the rate control scheme is implemented to adjust *cwnd*. When the third duplicate is received without the CW mark, Jersey concludes that the packet loss is caused by random packet error, so it enters the fast retransmit scheme without adjusting the window size. Under TCP Jersey, the TCP sender can set its congestion window to a more sensible value when congestion is detected. Moreover, the sender can differentiate congested losses from random error losses through the CW scheme. However, protocol modifications at intermediate nodes are necessary in order to implement the Jersey scheme.

- Explicit Congestion Notification [72]: Explicit Congestion Notification (ECN) combines additional Active Queue Management (AQM) mechanisms, such as Random Early Detection (RED) with the Internet framework. The main idea of ECN is that routers can detect network congestion and avoid buffer overflow events. Hence, the role of routers changes from packet dropping to congestion monitoring. Instead of dropping packets, the router marks congestion packets and sends them to the receiver. When these marking packets arrive at the receiver, congestion information is acknowledged to the sender and the congestion control mechanism is triggered. Since the congestion is monitored by intermediate nodes, its feedback is accurate.
- Wireless Explicit Congestion Notification: Wireless Explicit Congestion Notification (WECN) [90] extends ECN to wireless networks. The purpose of the WECN mechanism is to provide an effective way to separate congestion control mechanisms from retransmission strategies (i.e. packet loss or timeout events). If network congestion occurs in the wireless links, WECN messages are dispatched to the sender. Hence, the sender can detect the congestion situation through WECN messages. If a sender detects a packet loss but has not received any WECN messages, it means that the packet loss is due to link error. The sender does not need to enter the congestion control state. This method can precisely differentiate network errors coming from wireless or wired links. The drawback of WECN is that the modifications must be done simultaneously at both FH, MH and BS.

The end-to-end proposals maintain the basic TCP end-to-end semantics. The error detection approach considers its solution only at the modification on the sender side, so the influence on the current network is little, but a well-designed error detection mechanism is the key to success. The main advantage of the error notification approach is that the error estimation is more accurate than the error detection approach. However, the success of this approach relies on cooperation between sender and other nodes (BS or receiver), so flexibility in the future must be considered.

2.6.4 Summary of Wireless TCP Solutions

This section summarizes the observations about the split-connection and end-to-end proposals and concludes. Table 2.2 lists some important characteristics of split-connection and error notification proposals.

Table 2.2: Characteristics of split-connection and error notification proposals.

	I-TCP	Snoop	MTCP	ECN	ESBN	WECN	TCP Jersey
Preserves end-to-end semantics	No	Yes	No	Yes	No	Yes	Yes
Requires BS participation	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Changes code position	BS	BS	MH/BS	BS	FH MH/BS	FH MH/BS	FH/BS
Handles encrypted traffic	No	No	No	No	No	No	Yes
Handles asymmetric routing	No	No	No	No	No	No	Yes

Possible Solutions in Split-connection Proposals

According to [91], the implementation of Snoop is a good solution for recent developments of wireless TCP. Research results are encouraging, and it is an “invisible” optimization which does not affect either the client or the server, only the intermediate nodes. Besides, the end-to-end TCP semantic is maintained. However, Snoop does not work if the IP traffic is encrypted, unless the intermediate node shares a security association between the mobile device and its end-to-end peer. Two possible solutions have been proposed to solve this problem [91]. Firstly, a Snoop node is constructed as a party to the security association between the client and the server. Secondly, the IPSEC tunnelling mode is terminated at the Snoop intermediate node. However, these techniques require that users trust the intermediate node.

Main Issues in Split-connection Proposals

Five main concerns of split-connection proposals are considered, due to the required participation of base stations.

1. Overloading and optimum levels: Handoff loss is unavoidable when the MH is moving. During such periods, the control of MH is transferred from one BS to another. If the packet or retransmission information is stored in BS to speed up the following error recovery, all of the data need to be devolved on the next BS. Therefore, the time delay and additional usage of bandwidth for non-data transmission are still under observation. A well-designed buffer size at the BS is another issue for the success of this proposal.

2. Compatibility: Many dissimilar hardware and software systems are used at base stations. Thus, it is a challenge to have new compatible modifications for all systems.

3. Handling of encrypted traffic: Network security is an important topic of network connection, and data encryption is a necessary part of future transmission protocols. IP Security Protocol (IPSEC) is a popular IP encrypted protocol over the Internet. In the IPSEC solution, the whole IP payload is encrypted during transmission. It means that intermediate nodes cannot read any packet information. The proposal connects with the intermediate node based on extra information from the IP or TCP header, such as Snoop, I-TCP and MTCP, but this is unsuitable for transmission when the data has been encrypted.

4. Maintaining the semantic standard: Some protocols, such as I-TCP and ESBN, violate TCP end-to-end semantics, so that the original OSI-Layer model is contradicted.

5. Asymmetric routing: Split-connection proposals are not applicable to networks with asymmetric routing. Asymmetric routing means that the data packet and the acknowledged packet are routed from different paths. In this situation, the participation of intermediate nodes is useless.

Some end-to-end proposals, such as ECN and WECN, which are based on the participation of the BS, must also consider the effects described above.

Possible Solutions for End-to-end Proposals

The characteristic of “backwards compatible” are recommended to be considered into end-to-end proposals. Backward compatible refers to whether a newly introduced TCP proposal is compatible with the legacy TCP in the sense that it does not cause any detrimental effects to the legacy TCP and vice versa [85]. Since TCP has been widely used in the current live network, the adjusting at end hosts could have widespread effects on present Internet applications. Thus, “backwards compatible” is a better solution to restrict the effects on the current network framework.

Table 2.3: Characteristics of error detection proposals.

	TCPW	NCPLD	TCP Veno	Biaz	mBiaz	Spike	ZigZag	TCP Probing
Preserves end-to-end semantics	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Requires BS participation	No	No	No	No	No	No	No	No
Changes code position	FH	FH	FH	FH	FH	FH	FH	FH
Handles encrypted traffic	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Handles asymmetric routing	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

If the modification is only required at the sender host, it has more flexibility than the adaptations required at both sender and receiver hosts. Generally, the error notification approaches implement their modifications at both sender and receiver hosts. The receiver host is responsible for error notification when error is detected. The sender host is accountable for action response if any error notification is acknowledged from the receiver. Therefore, receiver mobile devices, such as mobile phones and personal digital assistants (PDA), must also apply new modifications to cooperate with the TCP sender to speed up the data transmission. In error detection approaches, the modification of TCP is usually and only implemented at sender host. The sender takes over both the error probing and error reaction. Thus, the TCP receiver is non-sensitive to the change on current TCP, but enjoys the performance improvement of data transmission. Moreover, the encrypted mechanism is maintained, because the modification of IP and TCP headers is unnecessary. Some significant features of the end-to-end proposals are listed in Table 2.3.

Main Concerns of End-to-end Proposals

Because the end-to-end proposals focus on their solutions at the TCP sender and receiver hosts, drawbacks from the participation of intermediate nodes are straightforwardly solved. Thus, the effects on the current Internet can be minimized. Besides, network security and data encryption topics would not be considered in the end-to-end proposals since the participation of intermediate nodes is unnecessary. As described above, end-to-end proposals that are backward compatible could be a good solution for enhancing the TCP performance in the heterogeneous network.

2.7 Simulation and Emulation Needs in TCP Research

In order to understand how TCP functions in the wireless network, network simulators are often used. Network simulators are useful tools for testing TCP performance in various network environments. The simulator can be used to understand the TCP behaviour, so further analysis and discussion can be taken. Simulator supporting the simulation components of wireless TCP have became critical as the result of the wide usage of wireless Internet access over recent years. Simulators can also reduce network hardware cost if a TCP study under a wide range topology is considered.

Based on the implementation type, network tools can be classified into two types: network simulators and network emulators, as explained in the following sections. The terms will be used throughout the thesis as the experimental over derived.

2.7.1 Network Simulators

Generally, a simulator is implemented as an application software, which supports a virtual network environment on which users can construct a virtual network topology. A single computer with a supporting operating system can simply create the simulator environment. High flexibility is the main feature of the simulator. It is usually used for the testing of new network protocols, with has high variation based on each simulated output. The advantages of using the simulator to evaluate the TCP performance are discussed below.

- The simulator is not limited by physical speed. For instance, a long term network demonstration could take a few days to gather the necessary data. Using the simulator, analytical information may also be collected in seconds.
- Simulation provides a means of testing TCP performance across “rare” networks, to which a researcher does not have easy access. [92]
- Simulators are not equipment extensions. A single computer can be used to run network simulations even for complex topologies. This feature is especially useful for simulating topology of end-to-end protocols, such as TCP and UDP, in which middle topologies between two hosts might not easily be controlled.
- A graphic user interface (GUI) is supported by many simulators. GUI can help users to look inside complex network structures. For instance, the TCP *ssthresh* and TCP *cwnd* are two factors that could affect TCP performance. Through GUI, the adjustment of two factors becomes simple and users do not need to deeply understand the real network structure.

Three simulator considerations are discussed below:

- Simulation is just "close to" the real network. A number of simulators, such as Network Simulator 2 (ns2), have been designed with an "abstract concept" for TCP implementation. The "abstract concept" involves only the main features of TCP, but ignores some TCP parts which must be used in the real network environment. For instance, in ns2, the TCP checksum algorithm or urgent data transfer on both sender and receiver sides are ignored by ns2. Besides, ns2 does not support a dynamic window advertisement. Moreover, there is no SYN/FIN connection establishement/teardown during TCP three-way handshaking. The most important point is that no real data is ever transferred. The "abstract concept" may be good enough to analyze TCP. But, the simulation results may not match those which are tested in the real network.

- Compatibility might cause the reliability issues. Customized modules are allowed to be deployed in those simulators. These modules might be developed by the users for some specific research purposes. The new modules might not be precisely examined, unlike those functions supposed by the simulators. Compatibility is not guaranteed. Afterwards, fault simulation might be produced due to the possible carelessness design and the faulty might not be found easily. Hence, care should be taken with the simulation results when any architecture is changed from the original simulator [92].

- Only partial network events can be "modelled" in the simulators, but no actual interactions are taken. However, in the real system, network is not the only factor, which affects network performance. For instance, network input/output (I/O) and file read/write(R/W) must fully cooperate with OS. For this reason, an OS might act as an important factor when the TCP throughput is studied. However, network simulators are usually implemented as an software application, so non-network events are hardly to be considered. Hence, the simulation accuracy could be highly improved if non-network factors can be included in the simulation.

In [93], some helpful rules for researcher to test the accuracy of their simulation are also discussed.

2.7.2 Network Emulator

In contrast with the simulator, the network emulator emulates the network which connects end systems (e.g. computers), not the end systems themselves. Emulation differs from simulation because a network emulator appears to be a network. Thus, end systems can be attached to the emulator and will behave as if they are attached to a network. Network emulators are useful for the observation of interaction with network devices and network environments since emulators are part of end systems. Some simulator drawbacks can naturally be solved owing the features of emulators. Using TCP as an example, a TCP emulator runs in a real network environment, so a "true" TCP connection must be established and a “real” data flow must be generated. The main drawback of the emulator is less flexibility. The emulator environment must be supported by a number of network devices, so extra cost and space are necessary.

2.8 A Survey of Executed Network Simulators and Emulators

Two techniques are usually used to construct a simulator; one using the “conceptual network protocol” [94, 95] and the other reusing “the real network kernel” of the operating system (OS) [23, 24, 25, 96, 97]. Simulators designed with a conceptual protocol are usually implemented as stand alone programs just like a network application. They are easy to set up and integrate. The main drawback of this kind of simulator is that the interaction and behaviour of a protocol over the operating system and the network device cannot be tested. Compared to a simulator constructed with a conceptual protocol, it is easier to observe the network behaviour in the real network in the simulator designed with the“real network kernel”. However, this kind of simulator is OS or OS kernel version specific.

2.8.1 Network Simulators

Network simulators can be classified into two types [98], as follows

- Real-time simulators set up the virtual topologies and traffic flows in real or scaled timeframes. The major advantage is that simulators can interact with the real network infrastructure and traffic flow. Simulators, such as Real network simulator [99] and ENTRAPID, are in this category. The drawbacks of the simulators are less flexibility and potential for extension than off-line simulators.

- Off-line simulators are designed in a virtual timescale. It means that the simulation time is not linearly related to real time but executes in the form of scheduled tasks. Compared with real-time simulators, off-line simulators are highly configurable and extensible. Simulators, such as ns2, ns3, OMNeT++ and OPNET belong to this category.

2.8.2 Real-time Simulators

The REAL Network Simulator

The objective of REAL [100] is to support the study of dynamic network behaviour in flow and congestion control schemes in packet-switched data networks. REAL provides around 30 modules which emulate the actions of several flow control protocols (such as TCP), and 5 research scheduling disciplines. REAL allows new modules to be added to the system with little effort. REAL also provides the source code, so that users can modify the simulator for their own purposes.

The ENTRAPID Network Simulator

The ENTRAPID [99] simulator introduces a new prototype using visualized networking kernels. Based on this prototype, traffic can re-enter variants of the standard BSD network stack through multiple instances. In this way, ENTRAPID provides a flexible network simulation through the network stack visualization in the user space. Compared with real kernel implementation, it increases the cost of overall performance at the simulation terminal.

2.8.3 Off-line Simulators

Network Simulator 2

The Network Simulator 2 (ns2) [94] is an object oriented network simulator. ns2 can simulate existing network protocols, such as TCP, routing and multicast protocols, over wired and wireless networks. ns2 is free and open source, so it is possible used to implement and test new network protocols and applications. ns2 can also be used as a limited-functionality emulator.

Network Simulator 3

The Network Simulator 3 (ns3) [101] is an open source simulator for networking development, education and research. It allows users to study Internet protocols and large-scale systems in a controlled environment. ns3 is written entirely in C++, including its user code, network protocols and topology scenarios. It uses a new design framework and is intended as an eventual replacement for the popular ns2 simulator. It tries to avoid some ns-2 deficiencies, such as interoperability and coupling between models, lack of memory management and debugging of split language objects, but it is not backwards-compatible with ns2. Although ns3 has only just been released (first version in 2008), it has already supported many novel simulated modules. For instance, ns3 supports simulation in peer-to-peer applications, IEEE 802.11 variants, IPv6 protocols, modern routing protocols and new network architectures.

OMNeT++ Network Simulator

OMNeT++ [95] is a public-source and open-architecture simulation environment. Its primary application area is the simulation of communication networks. But, new modules in the field of Internet simulations (IP,IPv6 etc), mobility and ad-hoc simulations have also been supported. OMNeT++ supports strong integrated GUI environment for specification and management of simulation scenarios.

OPNET

OPNET [102] software is not a free network simulator, and a legal licence is required for it. OPNET can be used for the virtual simulation of many current types of network technologies and protocols. It has an efficient simulation engine and users can modify memory utilization to allow complex simulations to be accomplished in a short time. It provides users with the ability to modify network parameters and see the effect of changes without extended hardware and software set-up.

OPNET uses an object-oriented modelling approach, and a graphical user interface (GUI) is supported. The OPNET model uses codes similar to C++. They are compiled and executed in the same way, so that a user who is proficient in C++ can control very detailed parts of the OPNET model. The main features of OPNET are described below:

1. It is designed in hierarchical network models, so the model can be nested within layers.
2. It uses object-oriented modelling, so can be referenced and used as a logical extension of object concepts.
3. Multiple network scenarios can be simulated and compared at the same time.
4. Simulation outputs can be analyzed using OPNET built-in graphing tools.

One main feature of OPNET is that it is interactive with a user using the simulation tool. The user can ask various ’What-if’ questions. For instance, the user can ask: ’What if I added four more client stations to the existing network?’ or ’Will the wireless signal be strong enough at this distance?’ These types of questions can be analyzed using the simulation tool without the need to purchase hardware. The function is also useful to estimate demand for hardware and software when the user is constructing a network topology.

The Harvard TCP/IP Network Simulator

The Harvard TCP/IP network simulator (HTCPIP) [96] reuses the TCP codes of the FreeBSD kernel without developing its own TCP modules. The main concept of HTCPIP is to build up the multiple virtual channels and virtual nodes in one host. Then, a mechanism named “private IP address”, is used to communicate with created virtual channels and nodes. In this way, simulated traffic can be conducted in-out of the FreeBSD TCP/IP stacks multiple times. The main advantage of HTCPIP is that a real-life BSD TCP code is executed, so the simulated interactions with OS and network devices are easily observed.

2.8.4 Summary of Simulators

Real-time simulators have a high relationship with OS network interfaces. By reusing the OS network code, simulator development time is minimized. Besides, the simulation results are highly realistic, since real network codes are used. The drawbacks of real-time simulators are less flexibility and extensibility compared with off-line network simulators. In off-line simulator environment, general objective simulations are easily executed. But, discrepancies between simulation conditions and real network situation are the main concern, since some interactions with OS cannot be considered, due to the designed structures of off-line simulators.

2.8.5 Network Emulators

Several different types of emulators have been presented. NIST Net [21] has been implemented as a Linux module and hooked inside the Linux kernel, which works as a multi-router. It is used to emulate performance dynamics on real IP packets passing through the Linux-based router. Seawind [103], focusing on wireless emulation, has been used on the simulation of wireless protocols, such as GPRS and Universal Mobile Telecommunication System (UMTS). Dummynet [23], Hitbox pseudo-devices [24] and x-SIM [25] are different network kernel extensions that intercept packets via real network stacks and emulate the network traffic. Mahrenholz [104] has designed a new emulator combining the ns2 [94] with the real network for wireless emulation. The restriction of this emulator is that the conceptual wireless modules are implemented for wireless simulation. Real wireless features, such as quick handover and external jamming, might not be truthfully demonstrated in the simulated outputs, and additionally, the emulator must construct each wired node on an individual processor, so large arrays of processors must be used. IMUNES [105] has designed a framework in which multiple virtual nodes are simulated in one FreeBSD processor. However, it is designed only for wired network emulation.

2.9 New Simulator for the TCP/IP Research

A well-designed network simulator could help future network research. But, the complex constructions of live network cannot not easily be simulated into one simulator. Thus, each proposed simulator has the simulation restriction in some topic. Therefore, we will propose a new prototype of TCP/IP network simulator. The simulator will focus on only TCP simulation functions but has some advantages that have not yet discussed by proposed simulators, as described above. The new prototype will combine the advantages of the real-time and off-line simulators. Chapter 3 will present a TCP/IP simulator, LTCP. Chapter 4 will present a TCP/IP emulator, the extension from LTCP. A simple and economic framework is the main advantage of the simulator and emulator. We will detail discuss in the next chapter.

2.10 Summary

This chapter sets the stage for the rest of the thesis by detailing TCP development and highlighting the weakness of TCP extended to the heterogeneous network. The basic TCP congestion control mechanisms will be further discussed and extended in the following chapters.

Some split-connection proposals can enhance TCP performance when TCP is extended to the heterogeneous network. However, the participation of base stations restricts development in the future. However, these considerations can be straightforwardly solved in the end-to-end proposals. If the TCP modification (which must be backwards compatible) is only configured on the sender side, the intermediate nodes and the receiver are not affected, and the cost of corresponding Internet modification can be limited. Therefore, end-to-end proposals (especially the sender side solution) will be addressed.

Many well-known TCP conclusions have been demonstrated by using network simulators. A flexible, user friendly and realistic TCP network simulator could contribute to future TCP research. In the following chapters, we show the results of an evaluation of the appropriate various of TCP congestion control for a heterogeneous environment.

Chapter 3

A TCP/IP Network Simulator

This chapter presents an easy extendable and realistic Linux TCP/IP network simulator (LTCP). LTCP uses real-life TCP/IP stacks to generate simulation results.

A network packet travelling within many different hosts can be actually simulated based on the use of the Linux TCP/IP stack on a particular node that has LTCP enabled. It also enables simulations of network topologies in real world systems. As a real Linux TCP/IP stack is used, the simulation environment is close to the real network.

Linux supported TCP algorithms are ready to be used in LTCP without additional development. For instance, TCP Reno, NewReno and Selective Acknowledgement (SACK), are originally applied into Linux TCP/IP stacks. It means that these TCP algorithms can naturally be simulated in LTCP. Therefore, two advantages apply. Firstly, every TCP algorithm, which has a Linux version of implementation, can be simulated in LTCP, so the simulation results are reliable. Secondly, if new TCP algorithms are developed in Linux TCP/IP stacks, these TCP algorithms can also be used in LTCP, so the extension of LTCP is simultaneous with the development of Linux TCP/IP stacks.

The Linux network interface and utilities are naturally included as tools for LTCP. For instance, "ftp" can be used to transfer real data from servers to receivers, so the packet flow is generated and then related statistics can be gathered. "ping" can be used to gather round-trip time between a server and a receiver. Standard network utilities like "ifconfig" or "route" can be used to set up the simulated network topology. In addition, "packet size" and "queue size" can be adjusted through the use of a Linux network interface. All of these features reduce the user learning time, because users operate the network tools over the real network in their daily life. The "develop time" of LTCP is also reduced. The solution by its nature will enable testing of such applications without alteration to the application code.

This chapter details the structure of LTCP. It starts with an introduction to the methodology and architecture of LTCP. The second section is a description of how the methodology was implemented under Linux. The following section displays the initial configuration of the simulator. The last section is a conclusion with the experimental results and discussions of simulations.

3.1 LTCP Architecture

LTCP architecture is discussed in this section. A detailed description of “LTCP information” can be found in Appendix A. Generally, a TCP/IP simulator always involves four fundamental functions:

- scenario generator: The scenario refers to the simulated network topology. In a TCP/IP simulator, a well-defined scenario generator can help users easily create different network topologies to observe TCP behaviours in different conditions.
- traffic generator: The traffic generator is used to create and inject data flow into defined simulation topology.
- TCP/IP architecture: It is the main part of the TCP/IP simulator. The TCP/IP architecture must simulate most TCP behaviours as they are in the real network. It includes a routing mechanism, which behaves as a real Internet TCP/IP layer.
- analytic tools: The analytic tools can help users easily study and analyze the TCP behaviour from complex simulation outputs.

LTCP uses the concept of “integration” instead of “creation” to accomplish these four functions. “Creation” means that all functions of the TCP/IP network will be individually developed. “Integration” means that the existing Linux TCP/IP stacks and network modules are considered as being assembled in the LTCP. Thus, consideration of LTCP only relates to the method for combining existing Linux TCP/IP architecture into the TCP/IP simulator. We will describe how to build up LTCP using the concept of “integration” in the following sections.

3.1.1 Creation of a Virtual Network Topology

A topology is a scenario of simulation. To construct a network topology is always the first step in making network simulations. A basic simulated topology must include at least node numbers and link situations. The node numbers indicate how many nodes will be involved in the simulation topology. Each pair of nodes must be connected with at least one link (the wired network). The link includes information about bandwidth, propagation delay, traffic direction and error state. When the simulation starts, all information supported in the network topology acts as the “input parameters” for use in the simulator.

A “scenario table” is used in LTCP to organize the user defined network topology. To be precise, to construct a simple link topology that has four nodes and six links, as shown in Figure 3.1, a user simply fills in the appropriate parameters in each field of the “scenario table”, as shown in Table 3.1.

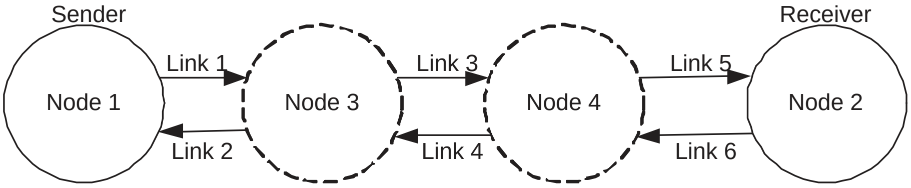


Figure 3.1: A network topology of four nodes and six links.

The first row means that link 1 is outgoing from node 1 and finishes at node 3. The bandwidth of link 1 is 10Mbps and propagation delay is 7 millisecond (ms). No packet loss occurs on link 1. Similarly, link 3 is outgoing from node 3 and finishes at node 4, and its bandwidth, propagation delay and packet loss rate are 10Mbps, 7ms and 0.1%. The packet loss rate will be detailed in Section 3.1.7. Through the "scenario table", different types of network topologies and link parameters can be integrated for use of LTCP. Table 3.1 will be retained for use in examples throughout this chapter. The values of bandwidth and delay are kept the same, in order to check the timings of the test applications.

Table 3.1: The "scenario table" used to construct a network topology.

Link No.	Source Node_No.	Destination Node_No.	Bandwidth (Mbps)	Delay (ms)	Packet Loss Rate %
1	1	3	10	7	0
2	3	1	10	7	0
3	3	4	10	7	0.1
4	4	3	10	7	0
5	4	2	10	7	0
6	2	4	10	7	0

3.1.2 Defining the Private IP Address for Each Virtual Node

In the real network, each network node has at least one IP address associated with its network interface. For this reason, one IP address is also associated with each node in LTCP. When the network topology has been created in LTCP, IP addresses are associated with nodes in this topology. The "private IP address" mechanism proposed in [96] is implemented here to associate the IP address in the node, defined as: the address for a link from node X to node Y is:

```
IP address of node X based on this link is:

192.168.Link_No.X
```

```
IP address of node Y based on this link is:

192.168.Link_No.254
```

In Figure 3.1, for example, with link 1 outgoing from node 1 to node 3, according to the definition above, the IP address of node 1 is 192.168.1.1 and that of node 3 is 192.168.1.254[1]. Similarly, with link 3 outgoing from node 3 to node 4, the IP address of node 3 is 192.168.3.3 and that of node 4 is 192.168.3.254. Thus, each sender and receiver node will have two IP addresses and each internal node (router) will have four IP addresses associated with it. The IP address end of number 254 will not be used in the system. Figure 3.2 shows the test configuration labelled with its IP addresses. Even though twelve IP addresses are associated with four nodes, only six addresses (bold numbers) are used in the simulator.

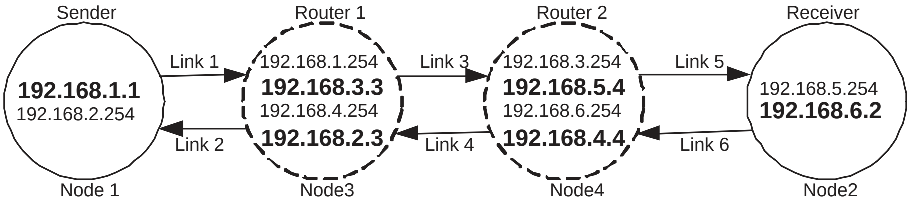


Figure 3.2: The conceptual view of the IP configuration is shown in four nodes after the definition of the "Private IP addresses".

3.1.3 Defining the Virtual Link Between Two Virtual Nodes

The network link between two nodes is another important factor of network topology. Linux supported TUN/TAP devices [106] are used to create virtual links between each pair of virtual nodes. TUN/TAP device is the virtual network device, which functions almost as the real network devices. TUN/TAP provides packet reception and transmission for user space programs. It can be viewed as a simple Point-to-Point (P2P) device, which, instead of receiving packets from a physical medium, receives them from the user space program. Similarly, instead of injecting packets via physical media, the TUN/TAP device writes them to the user space program. The TUN and TAP devices differ only in the fact that the TUN device is used for IP frames, and TAP is used for Ethernet frames[2]. When a Linux user program opens directory /dev/net/tun, a corresponding tunX device (i.e. X is a user supporting number) is created and registered via the mapping driver. Similarly, the tunX device and all routes which depend on it will be deleted after a user program terminates the tunX interface.

[1] The addresses 192.168.X.X are the best practice addresses for private networks as recommended in RFC1918 and the number 254 is arbitrary, as it is not used.

[2] Only the TUN device is used in the simulator at this moment.

When TUN devices have been created, IP addresses of virtual nodes are used to configure each TUN between two nodes. Each TUN device acts as a P2P link for a one-way link between nodes. The IP configuration of each TUN device as a P2P link associates TUNi with link i, which connects source node X and destination node Y.

```
TUNi Local IP address = 192.168.i.X

TUNi Remote IP address = 192.168.i.254
```

Figure 3.3 shows the TUN devices for the test configuration.

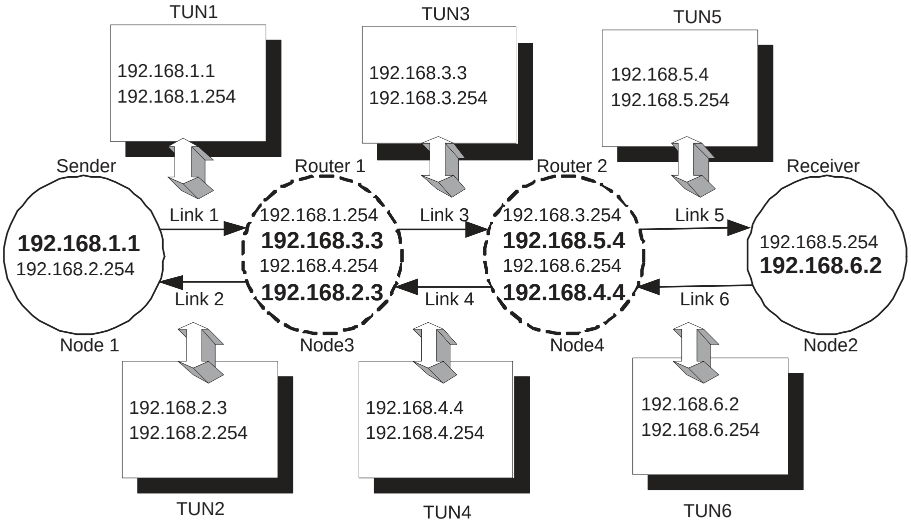


Figure 3.3: Conceptual view of IP configuration for each TUN device.

To be precise, according to the definition above, the TUN1's local IP address is 192.168.1.1 and the remote IP address is 192.168.1.254. Thus, TUN1 acts as a virtual link outgoing from node 1 and incoming to node 3. Similarly, the TUN6's local IP address is 192.168.6.2 and the remote IP address is 192.168.6.254, so the TUN6 acts as a virtual link outgoing from node 2 and incoming to node 4. Therefore, the association of links and nodes are connected together.

3.1.4 TCP/IP Protocol of the LTCP

A TCP/IP protocol architecture is the main part of the TCP/IP network simulator. LTCP uses the real Linux TCP/IP stack as the kernel of the TCP functions. As described in the last section, TUN/TAP devices are used as virtual link, so LTCP can simply redirect the traffic in-out of the Linux TCP/IP stack multiple times through the IP addresses, defining links and nodes, as described in the next section, so only one Linux PC is necessary to simulate a complex network topology.

Unlike the simulators proposed in [94] and [95], which developed the “conceptual TCP/IP code” to simulate TCP/IP behaviours, the real Linux TCP/IP stack and BSD Socket are retained for use in LTCP. The “conceptual TCP/IP code” means that some TCP functions are ignored during the simulation. Thus, a slight difference in TCP behaviours could exist between the simulator and the real network. However, the real TCP/IP is used in LTCP, so LTCP can naturally perform almost realistic TCP behaviours compared to those that use the “conceptual TCP/IP code”. There are three additional advantages:

1. Time does not need to be spent on developing the conceptualized TCP/IP code.

2. The Linux TCP/IP stack has been implemented in the real world for some time now, which means that the simulated statistics are reliable.

3. The current BSD Socket has supported many socket interfaces that can be used to connect from applications to BSD Socket directly. Since real BSD Socket has been applied for use with the current simulator, the socket interfaces can be used by applications connecting with the simulator without any need to alter the application code.

3.1.5 Routing Mechanisms in LTCP

The routing table is used to maintain correct routing information for traffic transmission. In LTCP, taking Figure 3.3 as an example, if we want to connect any two nodes with a path, serial settings of “route” must be ordered into the routing table, as presented in Table 3.2.

Even though additional routing setting can connect any two nodes with a path, an additional problem may occur. As shown in Table 3.2, IP address 192.168.3.3 has been set three times at tun1, tun4 and tun6. Similarly, IP address 192.168.6.2 has been set twice at tun3 and tun5. This situation might confuse the decision of Linux when it tries to route the traffic through the node with IP address 192.168.3.3 or 192.168.6.2. To solve this problem, the routing table could be maintained at each virtual node. However, this is not the best solution, as many changes to the routing source code would be needed inside the Linux. Thus, the “As-Seen-by-Node(i)” [96] is implemented as a routing mechanism in LTCP, so no routing code must be changed inside the Linux kernel.

Suppose that 192.168.Link_No.j is node j’s IP address. In the real world, node j’s IP address seen by other nodes should be the same, that is, 192.168.Link_No.j. But, under the definition of “As-Seen-by-Node(i)” algorithm, node j’s IP address seen by node i is 192.168.LIND_No.i. Similarly, node j’s IP address seen by node k is 192.168.LIND_No.k.

Table 3.2: A duplicate path setting occurs in the routing table.

```
route add 192.168.3.3 dev tun1
route add 192.168.5.4 dev tun1
route add 192.168.6.2 dev tun1
route add 192.168.4.4 dev tun1
route add 192.168.2.3 dev tun1

route add 192.168.1.1 dev tun2
route add 192.168.5.4 dev tun3
route add 192.168.6.2 dev tun3
route add 192.168.4.4 dev tun3

route add 192.168.2.3 dev tun4
route add 192.168.1.1 dev tun4
route add 192.168.3.3 dev tun4
route add 192.168.6.2 dev tun5

route add 192.168.4.4 dev tun6
route add 192.168.2.3 dev tun6
route add 192.168.1.1 dev tun6
route add 192.168.3.3 dev tun6
route add 192.168.5.4 dev tun6
```

In addition, some related routing information are set into the routing table to cooperate with "As-Seen-by-Node(i)", as presented in Table 3.3. The first block is the route setting for node 1. It means that any packet whose destination IP address is 192.168.2.1 will go through TUN1. Similarly, the second block is the setting for node 3, in which the packets' destination IP address 192.168.1.3 will go through TUN2, 192.168.4.3 will go through TUN3 and so on.

Take Figure 3.3 as an example. Suppose that packets transmit from node 1 (the sender) to node 2 (the receiver). The original definition of node 2's IP address should be 192.168.6.2, as described in last section. Under the mechanism of "As-Seen-by-Node(i)", the IP address of node 2 is seen as 192.168.6.1 by node 1, 192.168.6.3 by node 3 and 192.168.6.4 by node 4. After checking the routing table, packets from node 1 go through the virtual link 1 (TUN1) to node 3, virtual link 3 (TUN3) to node 4, virtual link 5 (TUN5) to node 2. When the packets arrive at node 2, its IP address seen by node 2 itself is 192.168.6.2. This "As-Seen-by-Node(i)" address is the same as its own IP address and the transmission finishes. Detailed processes are displayed in Figure 3.4. In this way, only one routing table is maintained in the kernel and packets can route in-out of the kernel many times without changing any kernel source codes. Besides, the setting of duplicate path in the routing table has been solved.

Table 3.3: The setting of packet routing paths in the routing table. The duplicate setting has been solved.

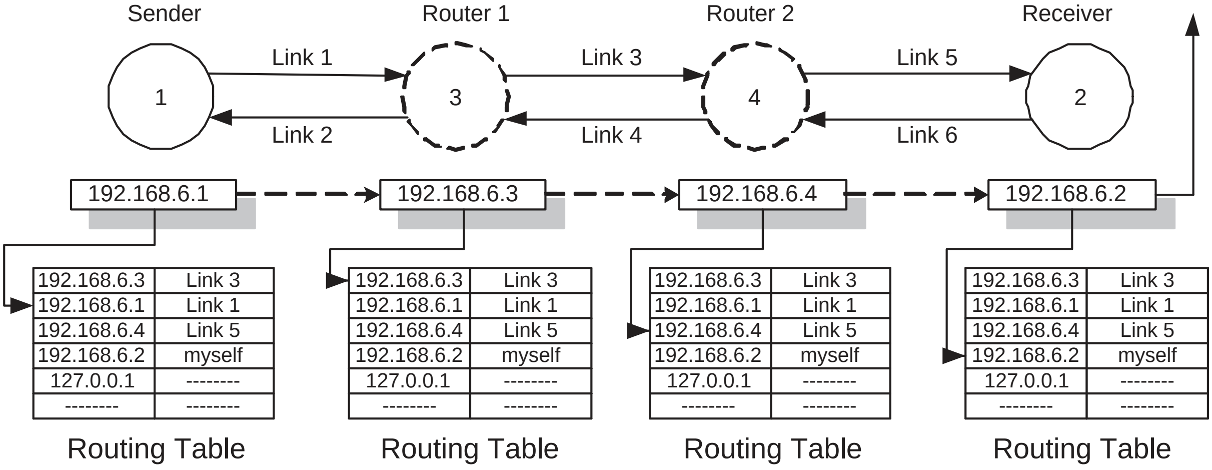
```
route add 192.168.2.1 dev tun1
route add 192.168.3.1 dev tun1
route add 192.168.4.1 dev tun1
route add 192.168.5.1 dev tun1
route add 192.168.6.1 dev tun1

route add 192.168.1.3 dev tun2
route add 192.168.4.3 dev tun3
route add 192.168.5.3 dev tun3
route add 192.168.6.3 dev tun3

route add 192.168.1.4 dev tun4
route add 192.168.2.4 dev tun4
route add 192.168.3.4 dev tun4
route add 192.168.6.4 dev tun5

route add 192.168.1.2 dev tun6
route add 192.168.2.2 dev tun6
route add 192.168.3.2 dev tun6
route add 192.168.4.2 dev tun6
route add 192.168.5.2 dev tun6
```

3.1.6 The Generation of Simulation Traffic

LTCP uses realistic network traffic instead of virtual traffic for data analysis. Unlike simulators [94] and [95], no real data is ever transferred during the TCP simulation. Based on the structure of LTCP, most Linux network applications can be operated to create network traffic. For instance, it is simple to construct a “ftp server” on the simulated server node using a “ftp client” application to set up a TCP connection between the server and the client. Thus, the data transfer from the server to the client is used as our simulated traffic. The experimental outputs are presented in Section 3.3.3.

3.1.7 User Space Event Scheduler

LTCP supports an event scheduler program in user space to maintain time consistency and simulate traffic states. Four functions are included: 1. the maintenance of a virtual timer, 2. packet catch, store and redirect, 3. delay simulation of virtual link, 4. packet drop generator.



192.168.6.3	Link 3
192.168.6.1	Link 1
192.168.6.4	Link 5
192.168.6.2	myself
127.0.0.1	--------
--------	--------

Routing Table

192.168.6.3	Link 3
192.168.6.1	Link 1
192.168.6.4	Link 5
192.168.6.2	myself
127.0.0.1	--------
--------	--------

Routing Table

192.168.6.3	Link 3
192.168.6.1	Link 1
192.168.6.4	Link 5
192.168.6.2	myself
127.0.0.1	--------
--------	--------

Routing Table

192.168.6.3	Link 3
192.168.6.1	Link 1
192.168.6.4	Link 5
192.168.6.2	myself
127.0.0.1	--------
--------	--------

Routing Table

Figure 3.4: IP address remapping uses the definition of "As-Seen-by-Node(i)" algorithm in LTCP.

Virtual Timer Maintenance

During the simulation, LTCP maintains a virtual timer instead of using the Linux supported system timer. The advantage of using the virtual timer is that, by changing the tick granularity of the virtual timer, the simulation time can easily be speeded up or slowed down. For instance, the default value for tick granularity of the virtual timer is 100 nanoseconds. If a user wishes to speed up the simulation, this tick granularity can be set at 5 microseconds. On the other hand, it can be set at a smaller value, such as 20 nanoseconds, to achieve results of higher accuracy or for high speed network simulation. If the virtual timer is used, all LTCP events must referenced to the virtual timer. Therefore, network utilities, applications and TCP timer are referred from the system timer to the LTCP's virtual timer. The necessary changes are discussed in Section 3.2.

Packet Catch, Store and Redirect

After packets arrive at each TUN device, the user space program invokes the *read()* system call to gather and store them into user space buffers. Using the *write()* system call, packets are picked up from user buffers and restored to each TUN device. The delay simulation of each virtual link is accomplished between each *read()* and *write()* call, as described in the next paragraph.

Delay Simulation of Virtual Links

A packet suffers from four major types of delay when it travels from one node (host or router) to the subsequent node (host or router). The processing delay is the time used to test the packet's header and decide where to direct the packet's next destination. The queuing delay is the time that a packet stays in buffers and waits to be transmitted. Transmission delay or store-and-forward delay can be simply explained in terms of L/R, in which L is the length of the packet (bits) and R is the link transmission rate (Mbps). When a packet is injected into a link, it needs to propagate to the next node. The time required to propagate from the beginning of the link and arrive at the next node is the propagation delay. The processing delay and queuing delay in the simulator are the spend time packets travelling in-out of the Linux kernel. Since we are using the actual TCP/IP stack, hence, only the transmission delay and propagation delay need to be included in the simulator. The processing delay and store-and-forward delay are the real time that packets remain in the kernel, and depend on different machine features.

In LTCP, two timestamps are set at each *read()* and *write()* system call in the user space program, so the link delay can simply be simulated. As shown in Figure 3.5, the first *read()* event is related to a timestamp A, denoted as $T(A)$, while the following *write()* event relates to timestamp B, denoted as $T(B)$. The relationship between two timestamps is shown in equation 3.1.

$$T(B) = T(A) + propagationdelay + transmissiondelay \tag{3.1}$$

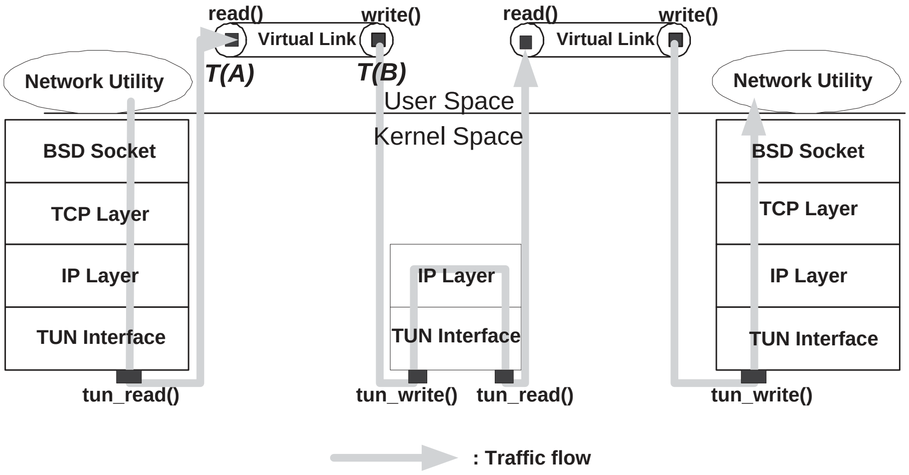


Figure 3.5: Virtual link simulation in the user program.

To be precise, when packets are "caught" by *read()* system call, LTCP records the current time $T(A)$ and stored packets in the user space buffers. Then, LTCP calculates the link delays. The propagation delay and transmission delay of the relative link are calculated through parameters defined in the "scenario table". For instance, if the total delay of link is 10ms, packets will be stored in the user space buffers for 10ms; then, packets will be restored from buffers through *write()* system call at time $T(A) + 10ms$, which is the $T(B)$ defined in equation 3.1. Then, packets will be injected into the TUN device and flows to the Linux

TCP/IP stack again. When the packets are stored in buffers, the pure packet first in first out (pfifo) algorithm has been implemented as queue discipline inside the simulated host. In pfifo queue discipline, a list of packets is maintained when the packet is enqueued and inserted at the tail of a list. When a packet needs to be sent out to the network, it is taken from the head of the list. The characteristic of pfifo queue discipline could cause lots of jitter, but the packets will stay in order.

Packet Drop Generator

Series of state switch are performed when a TCP connection suffers packet loss. For the inside observation of TCP behaviours, LTCP implements the Packet Drop Generator (PDG) to simulate the packet loss in the wired links. PDG simulates packet loss by dropping packets on the virtual link. When the packet is "caught" by the event scheduler program, the link PDG is triggered based on the "packet drop rate" defined in the "scenario table", to decide whether this packet is passed to the next node or is dropped. The PDG model uses the uniform distribution variable to determine dropped packets, where all packets have the same probability of being dropped, to simulate the scenario whereby packet loss occurs randomly due to the environment. Usually, in TCP simulations, packet loss during the TCP three-way handshaking step is avoided. The present PDG model uses a flag (S_flag), which needs to be set if SYN packets are not to be dropped. The following pseudo-code is implemented at each virtual link to simulate the packet drop situation.

```
random_value = uniform_dist(100);
get protocol type from pkt header;
if (protocol = TCP)
   if (S_flag) /*skip SYN packets*/
     if ((SYN is set at the packet) or (random_value > link_error_rate))
          pass packet to the next node;
     else
          drop the packet;
   else
   if (random_value < link_error_rate)
      drop the packet;
else /*non-TCP protocol*/
   if (random_value < link_error_rate)
      drop the packet;
```

Figure 3.6 presents a PDG demonstration over 50 seconds in which different packet drop rates are set at the specific link and the "ftp" application is used to generate the traffic. The high packet drop rate could cause high packet retransmission; hence, the lower transmission of packet sequence number is presented.

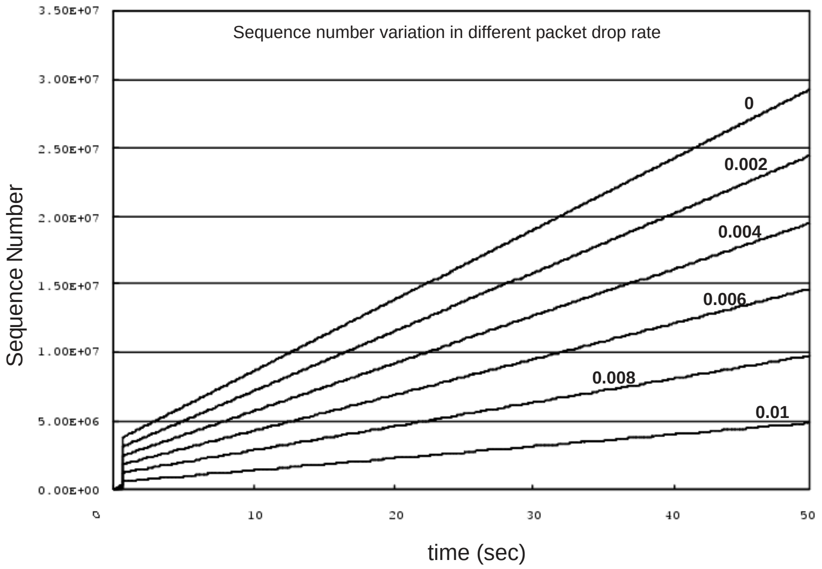


Figure 3.6: PDG implementation - the observation of sequence number based on SACK TCP with different packet drop rate.

In the wired network, serious packet losses could occur when queue is overflow in the middle nodes. Since PDG drop packets randomly, it is inappropriate to simulate the condition of queuing drop. LTCP can further integrated with Linux iproute [107] and netem [108] for advanced queuing discipline and drops.

3.2 LTCP Modifications in Kernel Space and User Space

Even though LTCP attempts to prevent any adjusting at Linux kernel for further flexibility, some modifications are unavoidable for LTCP to operate. Two modifications are made inside the Linux kernel: 1. consistent operation between virtual timer and TCP timer; 2. disabling the checksum functions; in the user space, reporting virtual time instead of system time as the simulated outputs is the main adjustment concern.

3.2.1 LTCP Modifications at Kernel Space

TCP Timers Based on Virtual Time

In a TCP connection, many functions rely on a correct time report. For instance, a TCP RTO event is triggered if a packet's relative ACK has not been reported for a period. Similarly, an activated TCP connection might be disconnected if no actions have been operated in this connection for a period. To accomplish these objects, many TCP events must precisely record their start time and end time. However, the original Linux TCP/IP state is designed to record the TCP event based on the system timer; but, LTCP refers to the virtual timer. Thus, the TCP timers must be referred to the virtual timer rather than the system timer. Otherwise, if the virtual timer is K times slower than the real timer, the TCP retransmit timer could expire prematurely at a time which is K times smaller than it should be.

Skip Tests of IP, UDP/TCP and TUN Checksum

LTCP must disable some checksum functions to avoid packet drop. During the simulation, LTCP changes the source and destination IP addresses at each node to redirect packets in-out of the Linux kernel; therefore, the checksums at the IP and the UDP/TCP layers must be disabled in the kernel. Otherwise, packets are dropped because of a checksum error and the simulation terminates. The checksum functions of TUN devices are also disabled for the same reason. The checksum at the TUN device can be skipped by setting the flag "TUNSETNOCSUM" at the device creation time.

3.2.2 LTCP Modifications at User Space

Time Reports from Network Utilities and Applications

In LTCP, network utilities and applications are referred to the virtual timer. Time records are important reference data for network simulation. As discussed in Section 3.1.6, most Linux based network utilities and applications can be used in LTCP. However, the system timer is usually used for them to report time records. As discussed above, LTCP maintains a virtual timer for simulation objectives. If network applications' time records refer to a system timer instead of a virtual timer, all recorded network events might be wrong. Thus, LTCP redirects the time references of the Linux tools from the system timer to the virtual timer.

3.3 The Simulator Configuration and Execution Output

Below are presented the execution results from the simulator. Table 3.1 and Figure 3.2 above are used as our example network topologies to illustrate how interface configuration and application execution are accomplished in the simulator. The LTCP source code can be found in the CD along with the thesis and the LTCP installation guide is described in A.

3.3.1 Setting Up and Configuring the TUN Device

Parameters defined in the "scenario table" (Table 3.1) are referred to input data for both P2P and routing configuration at each TUN device. A well-known Linux command *ifconfig* is used to do this operation. As the following six *ifconfig* show, each *ifconfig* command is executed to set a TUN device. The first *ifconfig* means that TUN1 is associated with a local IP address 192.168.1.1 and a remote IP address 192.168.1.254. After this, TUN1 is associated with a P2P link outgoing from node 1 and incoming to node 3. On the other hand, TUN2 will also act as a P2P link outgoing from node 3 and incoming to node 1.

It is important to note here that the IFF_NO_PI flag at each TUN device must be enabled at start time. Without setting this flag, extra meta packet information will be created and inserted into each packet by the TUN driver. When the user program tries to change the IP address of each packet, it will catch an error header data structure and result in error routing on a simulated network.

```
ifconfig tun1 192.168.1.1 pointopoint 192.168.1.254 netmask 255.255.255.0
ifconfig tun2 192.168.2.3 pointopoint 192.168.2.254 netmask 255.255.255.0
ifconfig tun3 192.168.3.3 pointopoint 192.168.3.254 netmask 255.255.255.0
ifconfig tun4 192.168.4.4 pointopoint 192.168.4.254 netmask 255.255.255.0
ifconfig tun5 192.168.5.4 pointopoint 192.168.5.254 netmask 255.255.255.0
ifconfig tun6 192.168.6.2 pointopoint 192.168.6.254 netmask 255.255.255.0
```

3.3.2 Setting Up the Routing Table for a Simulated Network

The commands "route" is used to insert additional routing information into the routing table. Table 3.4 demos the configuration of how to set a path for each two nodes in Figure 3.3.

3.3.3 Execution Results

Detection of Round-Trip Time

The utility "ping" is the most popular solution to test the round-trip time (RTT) between two nodes. It is also used to test whether the delay simulation of virtual links are correct or not. The first "ping" example below demonstrates estimation of the RTT between node 1 and node 2 (both of these are edge nodes). Since all the propagation delays are set as 7ms from links 1 to 6, total RTT between node 1 to node 2 should be 42ms. However, as shown in the following example, the first, fourth and fifth times RTT from "ping" reports are 43ms. The extra 1ms indicates the transmission and processing delay in the system.

Table 3.4: The setting of packet route paths in the routing table.

```
route add 192.168.2.1 dev tun1
route add 192.168.3.1 dev tun1
route add 192.168.4.1 dev tun1
route add 192.168.5.1 dev tun1
route add 192.168.6.1 dev tun1

route add 192.168.1.3 dev tun2
route add 192.168.4.3 dev tun3
route add 192.168.5.3 dev tun3
route add 192.168.6.3 dev tun3

route add 192.168.1.4 dev tun4
route add 192.168.2.4 dev tun4
route add 192.168.3.4 dev tun4
route add 192.168.6.4 dev tun5

route add 192.168.1.2 dev tun6
route add 192.168.2.2 dev tun6
route add 192.168.3.2 dev tun6
route add 192.168.4.2 dev tun6
route add 192.168.5.2 dev tun6
```

# ping 192.168.6.1 -w 5

```
PING 192.168.6.1 (192.168.6.1):  56 octets data
64 octets from 192.168.6.1:icmp_seq=0 ttl=62 time=43.0 ms
64 octets from 192.168.6.1:icmp_seq=1 ttl=62 time=42.0 ms
64 octets from 192.168.6.1:icmp_seq=2 ttl=62 time=42.0 ms
64 octets from 192.168.6.1:icmp_seq=3 ttl=62 time=43.0 ms
64 octets from 192.168.6.1:icmp_seq=4 ttl=62 time=43.0 ms


  --- 192.168.6.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 42.0/42.6/43.0 ms
```

In the real network system, “ping” is only executed on a host (edge node). However, under our simulator, “ping” can be used to report RTT between any two nodes (router to router). The following “ping” command is used to estimate the RTT between node 4 and node 3, both of which act as a router in the system.

# ping 192.168.4.3 -w 5

```
PING 192.168.4.3(192.168.4.3):56 octets data.
64 octets from 192.168.4.3:icmp_seq=0 ttl=64 time=14.0 ms
64 octets from 192.168.4.3:icmp_seq=1 ttl=64 time=15.0 ms
64 octets from 192.168.4.3:icmp_seq=2 ttl=64 time=14.0 ms
64 octets from 192.168.4.3:icmp_seq=3 ttl=64 time=15.0 ms
64 octets from 192.168.4.3:icmp_seq=4 ttl=64 time=14.0 ms

  --- 192.168.4.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 14.0/14.4/15.0 ms
```

ftp Connection between Two Nodes

The “ftp” connection between two nodes (i.e. client and server) is a simple way to generate traffic flow on the simulator. The following example explains that a ftp connection is created between node 1 (the client) and node 2 (the server). After the connection is established, the “get” command is used at this connection to get the file “testfile.ps” from the server. In this way, the traffic transmission is generated easily on each simulated network. Moreover, each virtual node is associated with a private port number in the system. For instance, node 2 has a port number 8003, as shown below. Hence, more than one ftp connection can be simulated simultaneously without any conflict.

```
# ftp -N 1 -S ftp2 192.168.6.1
clientNode No.is 1
serverNode No.is 2
server port number is 8003
Connected to 192.168.6.1.
220 kwl FTP server ready.
Name (192.168.6.1:root):  root
331 Password required for root.
Password:
230 User root logged in.
Using binary mode to transfer files.
ftp> get testfile.ps
local:  testfile.ps remote:  testfile.ps
200 PORT command successful.
150 Opening BINARY mode data connection for ‘testfile.ps’ (3036456 bytes).
226 Transfer complete.
3036456 bytes received in 3.2 secs (10.2e+02 Kbytes/sec)
```

The option "-N 1" is set to start the ftp client at node 1. Similarly, the option "-S ftp2" is used as follows to connect the ftp client to a server or to node 2. In addition, the above result can help confirm that the system can simulate 10 Mbps links with an 576 byte maximize segment size as we set in topology in Table 3.1. In the simulator, "ftp" can be used to generate the traffic between any two nodes (router to router). The following example shows that we have created a ftp connecting between node 3 (client) and node 4 (server). This capability is very useful for simulating that the network traffic need not come from edge hosts.

```
# ftp -N 3 -S ftp4 192.168.4.3
clientNode No.is 3
serverNode No.is 4
server port number is 8007
Connected to 192.168.4.3.
220 kwl FTP server ready.
Name (192.168.4.3:root):  root
331 Password required for root.
Password:
230 User root logged in.
Using binary mode to transfer files.
ftp> get testfile.ps
local:  testfile.ps remote:  testfile.ps
200 PORT command successful.
150 Opening BINARY mode data connection for 'testfile.ps' (3036456 bytes).
226 Transfer complete.
3036456 bytes received in 2.88 secs (1e+03 Kbytes/sec)
```

Display Routing Path between Two Nodes

The utility "traceroute" can test and display whether routing path matches the network topology in the simulator. The following example illustrates the "traceroute" output from node 1 to node 2.

```
# traceroute 192.168.6.1
traceroute to 192.168.6.1 (192.168.6.1), 30 hops max, 40 bytes packets
1 192.168.2.1 (192.168.2.1) 15.000 ms 15.000 ms 14.000 ms
2 192.168.4.1 (192.168.4.1) 28.000 ms 30.000 ms 28.000 ms
3 192.168.6.1 (192.168.6.1) 42.000 ms 42.000 ms 42.000 ms
```

The first output IP address 192.168.2.1 means that it is the outgoing address from node 3 to node 1 using the "As-Seen-by-Node(i)" algorithm. The same results are presented at the second and third lines. From this "traceroute" output, it may be understood that the routing path from node 1 to node 2 will go through node 3, node 4 and finally arrive at node 2.

Monitor Network Status

The utility "tcpdump" is used to monitor packet transmission on a network interface. It can be used directly to monitor network traffic on any TUN device under the simulator. The following example illustrates that we want to monitor packet transmission at TUN5 (link 5). The traffic flow is generated by the "ftp" command from node 1 to node 2 as mentioned before. Hence, 192.168.1.4 is the traffic source address (node 1) seen by node 4 and 192.168.6.4 is the destination address (node 2) seen by node 4.

```
# tcpdump -i tun5
tcpdump:  listening on tun5
10:37:27.465155 192.168.1.4.32776 > 192.168.6.4.ftp2:
```

3.3.4 Demonstration of Various TCP Implementations

LTCP can be used to evaluate the performance and observe the behaviour of different TCP versions. Three well-known TCP versions: NewReno, SACK and Vegas are tested for demonstration. In Linux, SACK is implemented based on TCP NewReno. SACK shares the same slow-start and additive increase algorithms with NewReno. The main difference between SACK with NewReno is dealing with the multiple packet losses in a single TCP window. TCP Vegas is supported by LTCP for Linux kernel 2.4.

Figure 3.7 shows the network topology. The sender and receiver are separated by two routers, 50ms propagation delay and a 2M bottleneck link. The drop-tail queuing discipline is used. No additional packet loss event is set between links, so the packet drop happens naturally from the queuing drop (buffer overflow). The Maximum Transmission Unit(MTU) is 1500 Bytes, the default setting of Ethernet topology in the Linux. "ftp" application is used to generate the traffic data. "tcpdump" utility is used to catch the TCP data from each TUN device. "tcptrace" utility [109] is used to analyze the output data from the "tcpdump". A detailed description of "tcptrace" can be found in Appendix B.

A good way to visualize the differences between NewReno, SACK and Vegas is to study sequence number versus time plots of their conversations. This section presents small tutorial examples. Later, Chapter 4 presents plots taken from live and emulated networks under various traffic loads.

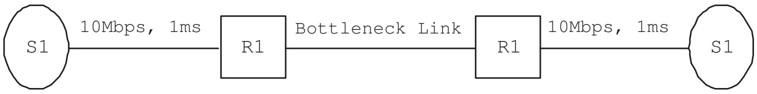


Figure 3.7: Network topology for simulations

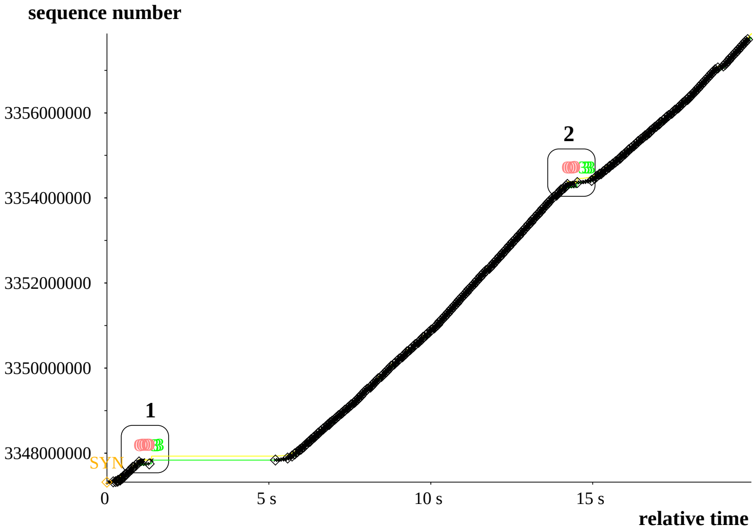


Figure 3.8: NewReno: sequence number vs. time

Experiment on TCP NewReno

TCP NewReno has been described in Section 2.3.6. Figure 3.8 plots sequence number versus time of 10MB transfers. Despite the lack of competing traffic, shortly after the NewReno transfers begin, as marked at label 1, it experiences a 4 second timeout because of the buffer overflow at the intermediate node. Hence, NewReno resets its *ssthresh* and adjusts the transmission state from exponential growth to linear growth. After the timeout event, NewReno's congestion avoidance algorithm avoids further packet drops until time 14s as marked at label 2. Eventually, even linear growth causes a drop, and we see inflection points besides the timeout, indicating that NewReno drops packets and halves its *cwnd* once during this 10MB transfer. Only the first drop causes a timeout; the fast recover algorithm successfully patches the second drop. Figure 3.9 and Figure 3.10 are inside looks at label 1 (left) and label 2 (right) individually.

Figure 3.9 focuses on the first few seconds of a NewReno transfer until the sender has been acknowledged the loss event by duplicated ACKs. **3** indicates that the received ACK packet was the triple duplicate ACK, commonly used as the threshold to trigger the fast retransmit and fast recovery algorithm. **O** presents packets received out of order. The line **A** tracks the receive window advertised from the other endpoint. Line **B** represent segments sent. The up and down arrows represent the sequence numbers of the last and first bytes

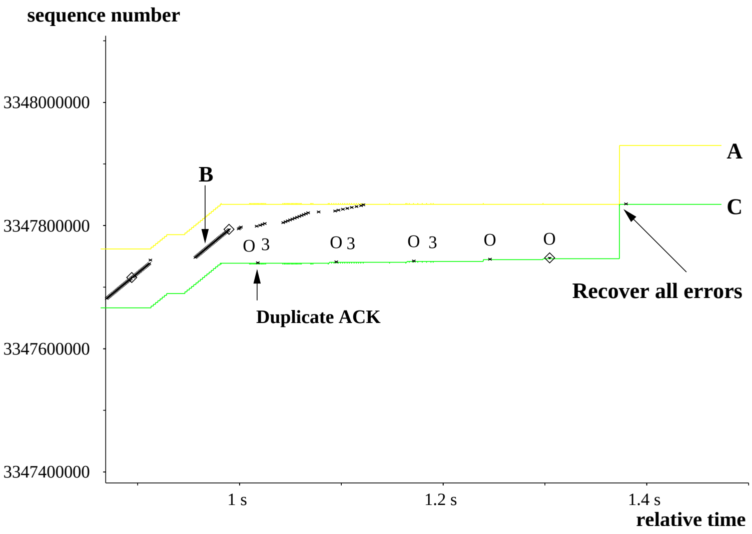


Figure 3.9: The inside look at label 1 of Figure 3.8

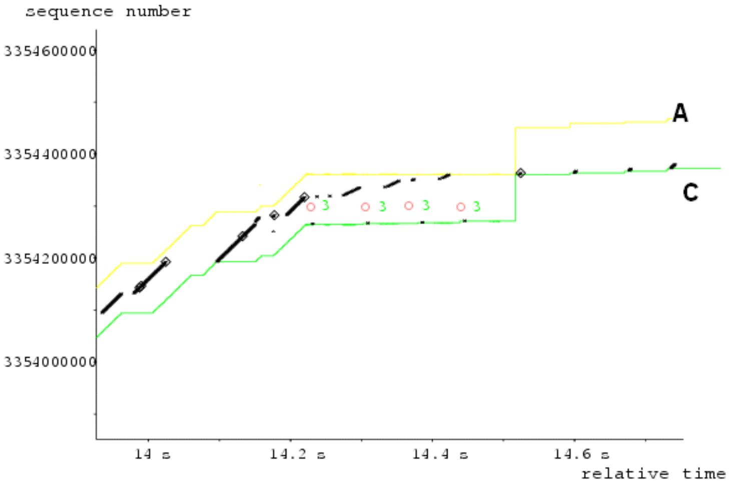


Figure 3.10: The inside look at label 2 of Figure 3.8

of the segment respectively. Line **C** keeps track of ACK values received from the other endpoint; and the little ticks on line **C** track the duplicate ACKs received. The line **A** and **C** will also appear in Figure 3.10 and Figure 3.13. At around 1s, the sender first receives an out-of-order ACK; after a few milliseconds, the sender receives three duplicate ACKs of this out-of-order packet and the fast retransmit algorithm is triggered. Between 1s to 1.4s, 5 out-of-order packets are noticed, and the fast retransmit algorithm is triggered three times to carry out the fast recovery algorithm. Although a new ACK for the lost packet eventually arrives (after around 1.4s), the gap between the highest transmitted segment and the highest acknowledged segment is so large that NewReno must wait for its retransmission timeout to expire. NewReno, faced with more than 4-5 drops in a single RTT, may occur the timeout event; depending on the size of *cwnd*.

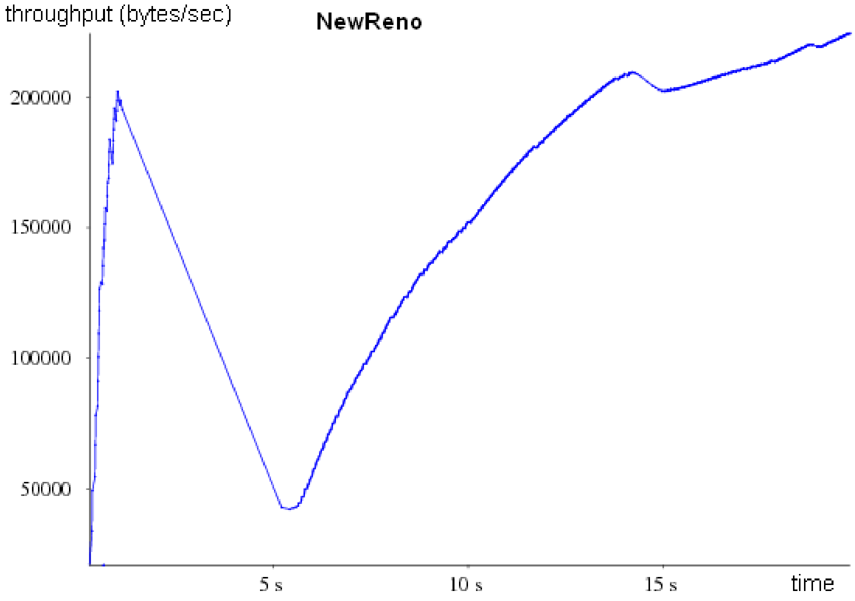


Figure 3.11: NewReno Simulation: throughput vs. time

Figure 3.10 focuses on the inside look at label 2 of Figure 3.8. Its situation is very similar to that in Figure 3.9. However, this time, the gap between the highest transmitted segment and the highest acknowledged segment is not too large; thus, the fast retransmit algorithm indeed recovers the loss event. Even though a similar loss event is presented, the effect of TCP throughput is different. Figure 3.11 shows the average throughput of the NewReno transmit. Between 1s to 5s, NewReno must enter a timeout event to cover the lost packets, so that the transmission of new packets is temporarily blocked, until all errors are recovered. The throughput goes down from 200,000 bytes/s to 40,000 bytes/s. At around 14s, the fast retransmit algorithm solves another lost event. However, there is only a slight effect to the average throughput. To summarize, the timeout event affects TCP throughput more seriously than the fast retransmit algorithm. Therefore, if the timeout event could be avoided or reduced from transmission of the TCP link, the TCP performance could be dramatically increased.

3.3.5 Experiment on TCP SACK

TCP SACK is described in Section 2.3.7. As shown in Figure 3.12, SACK's transmit condition is similar to NewReno, in which a packet drop happens because of buffer overflow, marked on labels 1, 2 and 3 separately. However, SACK retransmits the lost packet without experiencing the timeout event. This is because SACK can recover errors when multiple packets are dropped from one window of data. Figure 3.13 is the inside look of labels 1, as marked in Figure 3.12. The label 2 and 3 events marked in Figure 3.12 are similar to label 1, so only the event of label 1 is discussed below.

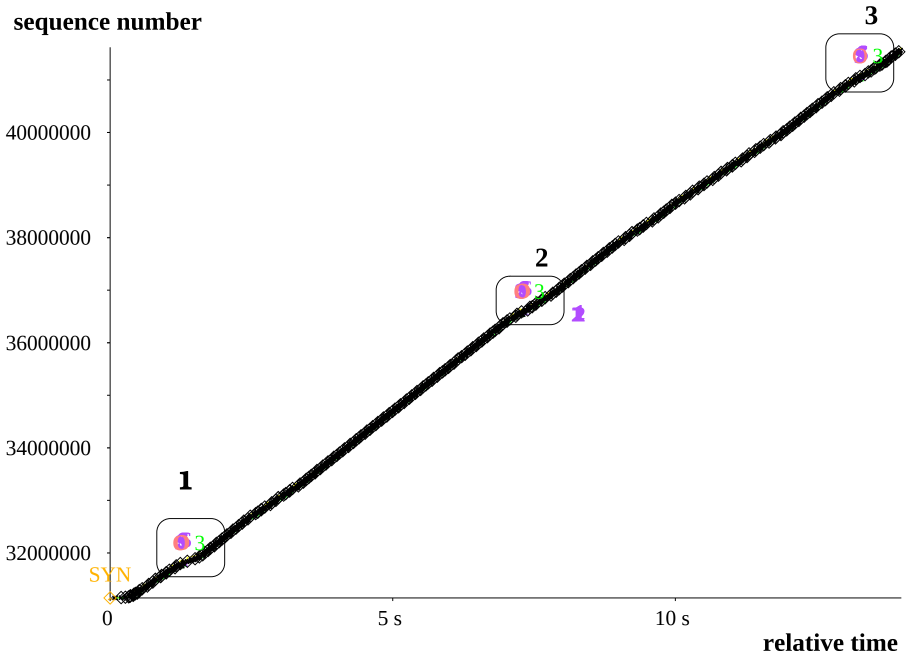


Figure 3.12: SACK Simulation: sequence number vs. time

Figure 3.13 focuses on the first few seconds of a SACK transfer, until the sender has been notified of loss events. **3**, **O**, **A** and **C** have the same meaning as described in last section. SACK blocks in ACK packets are represented as lines with an **S**. We use an example to demonstrate the proper behaviour of SACK generation by the TCP receiver and then describe Figure 3.13. Assume 4000 bytes are sent in 8 different TCP packets, each containing 500 data bytes. The first segment is dropped but the remaining 7 are received. The sequence number of first packet starts from 5000. In this case, relying purely on the cumulative acknowledgment scheme employed by the original TCP protocol, the receiver cannot say that it received bytes 5500 to 8999 successfully, but failed to receive the first packet, containing bytes 5000 to 8999. Thus the sender would then have to resend all 4,000 bytes (8 packets). It can lead to inefficiencies when packets are lost.

SACK allows the receiver to acknowledge discontinuous blocks of packets that are received correctly, in addition to the sequence number of the last contiguous byte received successively, as in the basic TCP acknowledgment. The acknowledgement can specify a number of SACK blocks, where each SACK block is conveyed by the starting and ending sequence numbers of a contiguous range that the receiver correctly received. In the example above, the receiver would send SACK with sequence numbers 5500 and 9000. The sender will thus retransmit only the first packet, bytes 5000 to 5499 (first lost packet). In SACK, upon receiving each of the last seven packets, the TCP receiver will return a TCP ACK segment that acknowledges sequence number 5000 and contains a SACK option specifying one block of queued data, as shown in Table 3.5.

- Left Edge of Block: This is the first sequence number of this block.

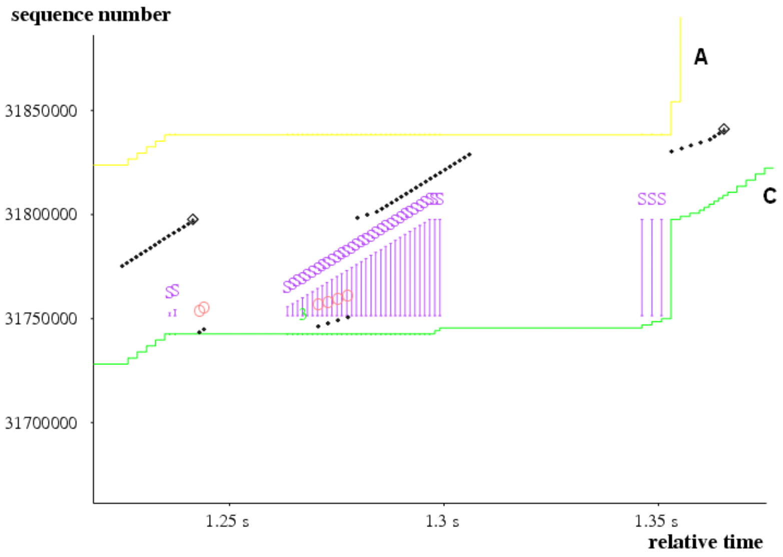


Figure 3.13: SACK Simulation: sequence number vs. time

Table 3.5: Example of SACK mechanism.

Triggering Segment	ACK	Left Edge Block	Right Edge Block
5000	(lost)		
5500	5000	5500	6000
6000	5000	5500	6500
6500	5000	5500	7000
7000	5000	5500	7500
7500	5000	5500	8000
8000	5000	5500	8500
8500	5000	5500	9000

- Right Edge of Block: This is the sequence number immediately following the last sequence number of this block. Each block represents received bytes of data that are contiguous and isolated.

Each SACK block reports a non-contiguous set of data that has been received and queued. The first block in a SACK is required to report the TCP receiver' s most recently received segment, and the additional SACK blocks repeat the most recently reported SACK blocks [41]. Then, the TCP receiver awaits the receipt of data (perhaps by means of retransmissions) to fill the gaps in sequence space between received blocks.

In Figure 3.13, at around 1.23s, the TCP receiver receives two out-of-order packets. Then, the receiver holds these two packets in the queue (with two **S** symbols) and waits for the previous packet (late arrive or loss) because the SACK is implemented. At around 1.24s, the previous packet arrives and fills the hole. The receiver then leaves the error type and returns to the normal transmission. Similarly, at around 1.27s, another out-of-order event is detected by the TCP receiver. But, the receiver waits the previous packets longer than the first time. 32 out-of-order packets are held into the queue between 1.27s and 1.3s. Until 1.36s, the previous packets arrive and the TCP leaves the error type. In SACK, the out-of-order packets can be queued and wait for the previous packets without falling into the RTO, so SACK and keeps in high *cwnd* and the effect in throughput is minimized.

Figure 3.14 shows the average throughput of SACK transmit. Even though the loss event is similar to that in NewReno, the simulation results show that SACK's retransmission strategy can effectively improve NewReno's fast recovery algorithm by avoiding a number of lost events falling into retransmission timeout, so the average throughput of SACK is enchained. Similar outputs were also discussed in [39, 52, 110]

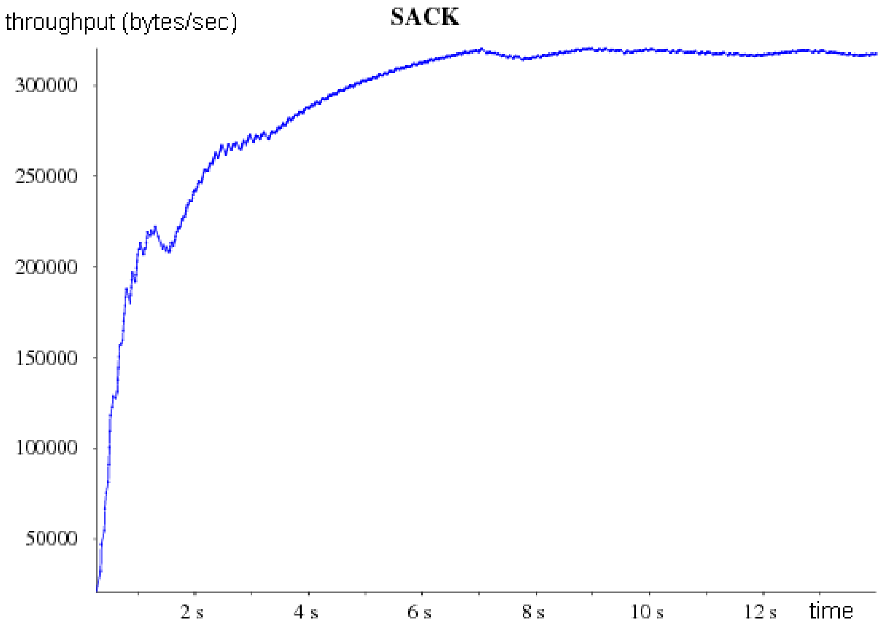


Figure 3.14: SACK Simulation: throughput vs. time

3.3.6 Implementation of Vegas Algorithm into Linux Kernel

LTCP also implements the source code of TCP Vegas into the Linux kernel 2.4. Unlike Reno, NewReno and SACK, TCP Vegas was not originally implemented into Linux kernel 2.2 or kernel 2.4.20[3], but, the idea of TCP Vegas has been widely implemented into a number of new TCP versions [111, 85] in the last few years. For further research into TCP and Vegas, plug in the Vegas algorithm into Linux kernel 2.4.20, so we can understand Vegas' behaviour using the simulator. This contributes to TCP research in the future.

[3] Note that TCP Vegas will be implemented into Linux kernel 2.6.

3.3.7 Experiment on Vegas

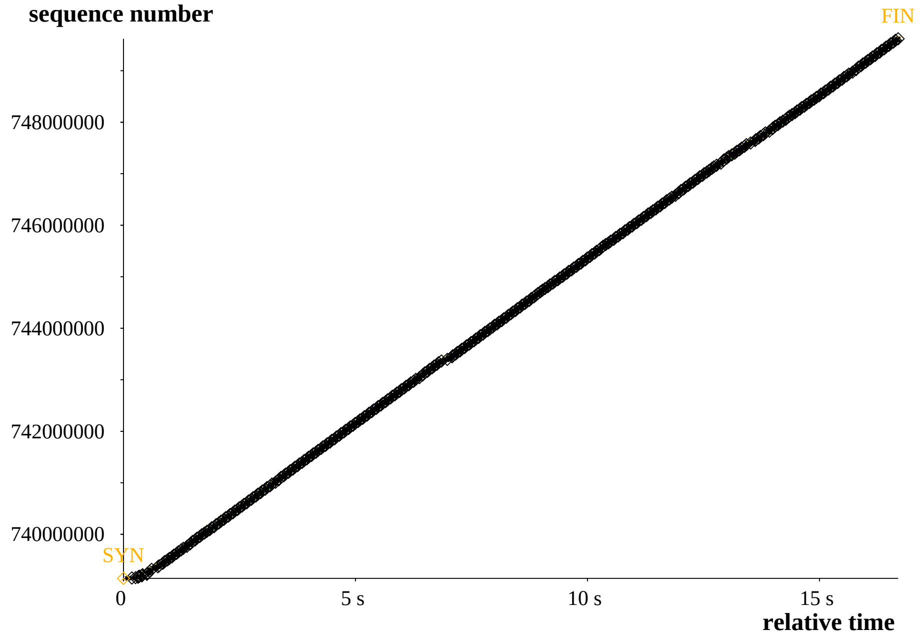


Figure 3.15: Vegas Simulation: sequence number vs. time

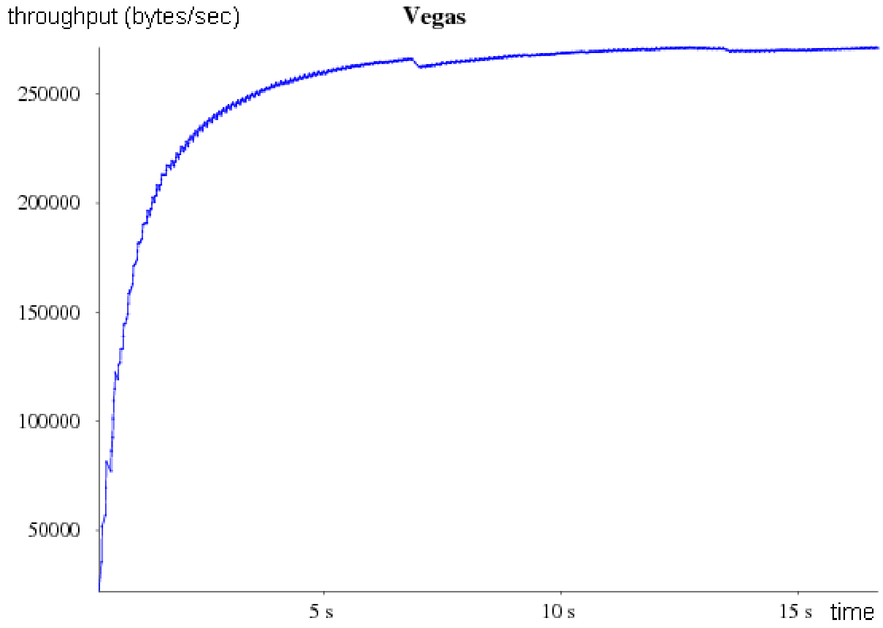


Figure 3.16: Vegas Simulation: throughput vs. time

TCP Vegas has been described in Section 2.4. As shown in Figure 3.15, the Vegas sender transmits entire test data without a single packet drop. This is because Vegas' congestion avoidance algorithm estimates the available link bandwidth and minimum propagation delay, and never increases its congestion window large enough to cause drops. On NewReno's initial drop, Vegas gains two to three seconds; for each of NewReno's three subsequent drops, Vegas gains a network RTT. Without the RTT event, the TCP throughput can be kept in a stable condition. Figure 3.16 shows the average throughput of Vegas transmit. Since no packet drop happens during transmission, Vegas' average throughput is smoother than NewReno's and SACK's.

3.3.8 Summary of TCP Simulation in LTCP

Figure 3.17 displays the average throughput of three TCP versions. In this simulated topology, NewReno experiences two retransmission events. At the first error event, NewReno enters a timeout event. In this period, NewReno suspends packet transmission until the sender retransmits all lost packets. Thus, the average throughput of NewReno drops seriously at this moment. At the second error event, NewReno enters the fast retransmit and fast recovery event and successfully retransmits all lost packets. Thus, the effect of average throughput is small.

Vegas uses estimate packet numbers in a queue to evaluate available link bandwidth, so it can "prevent" packet loss in advance. Thus, the average throughput of Vegas becomes "smooth". However, overestimation of the queue state could keep TCP *cwnd* in a low range, so the link bandwidth can not be used effectively. This phenomenon can be found in Figure 3.17. Vegas' throughput is maintained at around 250000 (Bytes/sec) after 5s and keeps this value until the end of simulation; that is to say, Vegas stops growing *cwnd* after 5s.

In simulation, even though three loss events are detected during SACK transmission, it is found that the average throughput of SACK is higher than Vegas' and NewReno's throughput. The main reason that SACK's throughput higher than NewReno's is that SACK strategy can improve on the NewReno retransmission algorithm by avoiding falling into retransmission timeout, so SACK can maintain *cwnd* at a high value and maintain the high throughput. In the simulated comparison of SACK and Vegas, SACK keeps increasing the *cwnd* even though there are packet loss events during transmission; that is to say, SACK can use link bandwidth more effectively than Vegas, so SACK has better average throughput than Vegas.

Several TCP experiments have been referred to in this section. It is clear that TCP behaviour can be simply simulated and observed in LTCP. The simulation outputs from the real TCP stack can support almost as "realistic" results as those comes from the real network environment.

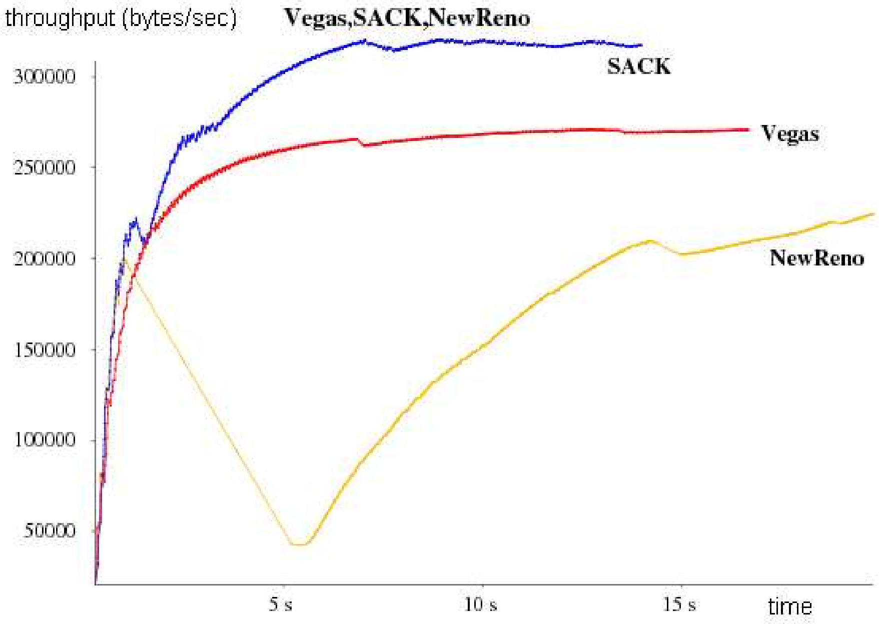


Figure 3.17: NewReno, Vegas, SACK: throughput vs. time

3.4 Summary Regarding LTCP

3.4.1 Simulation of Mesh Topology

In LTCP, each simulated link uses a TUN device as simulated link. The maximum number of links is limited by the maximum number of of TUN devices that a Linux system can support. The Linux supported TUN device is currently limited in 256. If the simulated links is more than 256, it is still possible to increase the Linux supported TUN device by modifying the kernel device system.

Currently, it is sufficient used 256 links to construct a complex simulated topology. In this section, we illustrate the configuration of LTCP for a topology that is more complex than the simple one used in Section 3.1 displayed in Figure 3.3. Figure 3.18 depicts the mesh topology discussed in this section.

TUN Network Device Configuration

The commands used to configure the twelve TUN devices for link 1, 2, .., and 12 of Figure 3.18 are shown below:

```
ifconfig tun1 192.168.1.1 pointopoint 192.168.1.254 netmask 255.255.255.0
ifconfig tun2 192.168.2.4 pointopoint 192.168.2.254 netmask 255.255.255.0
ifconfig tun3 192.168.3.3 pointopoint 192.168.3.254 netmask 255.255.255.0
```

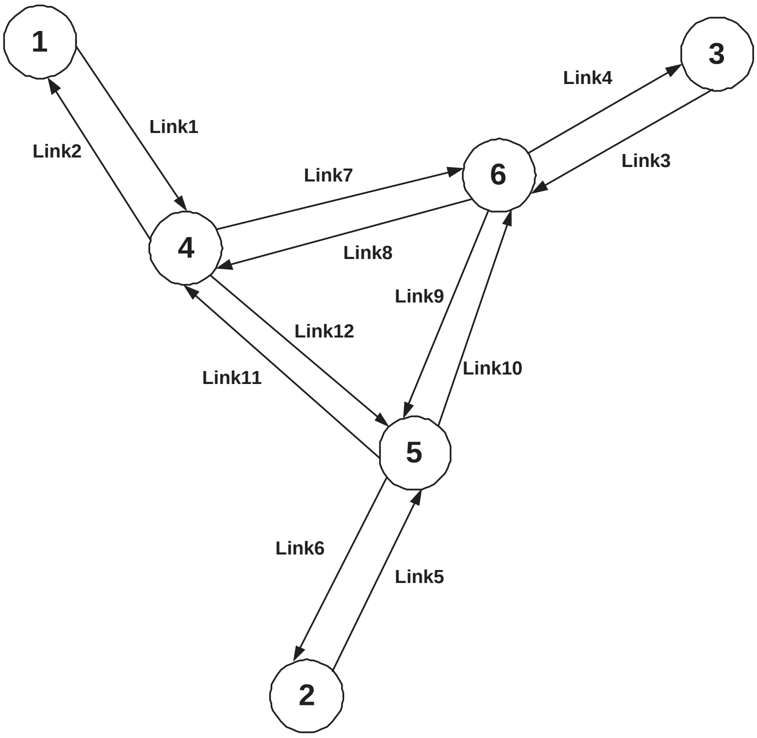


Figure 3.18: A mesh simulated network topology to illustrate configuration of the LTCP

```
ifconfig tun4 192.168.4.6 pointopoint 192.168.4.254 netmask 255.255.255.0
ifconfig tun5 192.168.5.2 pointopoint 192.168.5.254 netmask 255.255.255.0
ifconfig tun6 192.168.6.5 pointopoint 192.168.6.254 netmask 255.255.255.0
ifconfig tun7 192.168.7.4 pointopoint 192.168.7.254 netmask 255.255.255.0
ifconfig tun8 192.168.8.6 pointopoint 192.168.8.254 netmask 255.255.255.0
ifconfig tun9 192.168.9.6 pointopoint 192.168.9.254 netmask 255.255.255.0
ifconfig tun10 192.168.10.5 pointopoint 192.168.10.254 netmask 255.255.255.0
ifconfig tun11 192.168.11.5 pointopoint 192.168.11.254 netmask 255.255.255.0
ifconfig tun12 192.168.12.4 pointopoint 192.168.12.254 netmask 255.255.255.0
```

Route Configuration

The commands used to configure the routes for the simulated topology of Figure 3.18 are shown as follows:

```
route add 192.168.2.1 dev tun1
route add 192.168.3.1 dev tun1
route add 192.168.4.1 dev tun1
route add 192.168.5.1 dev tun1
route add 192.168.6.1 dev tun1
route add 192.168.7.1 dev tun1
route add 192.168.8.1 dev tun1
route add 192.168.9.1 dev tun1
route add 192.168.10.1 dev tun1
route add 192.168.11.1 dev tun1
route add 192.168.12.1 dev tun1

route add 192.168.1.2 dev tun5
route add 192.168.2.2 dev tun5
route add 192.168.3.2 dev tun5
route add 192.168.4.2 dev tun5
route add 192.168.5.2 dev tun5
route add 192.168.6.2 dev tun5
route add 192.168.7.2 dev tun5
route add 192.168.8.2 dev tun5
route add 192.168.9.2 dev tun5
route add 192.168.10.2 dev tun5
route add 192.168.11.2 dev tun5
route add 192.168.12.2 dev tun5

route add 192.168.1.3 dev tun3
route add 192.168.2.3 dev tun3
route add 192.168.3.3 dev tun3
route add 192.168.4.3 dev tun3
route add 192.168.5.3 dev tun3
route add 192.168.6.3 dev tun3
route add 192.168.7.3 dev tun3
route add 192.168.8.3 dev tun3
route add 192.168.9.3 dev tun3
route add 192.168.10.3 dev tun3
route add 192.168.11.3 dev tun3
route add 192.168.12.3 dev tun3

route add 192.168.1.3 dev tun2
route add 192.168.3.3 dev tun7
route add 192.168.4.3 dev tun7
route add 192.168.5.3 dev tun12
route add 192.168.6.3 dev tun12
route add 192.168.8.3 dev tun7
route add 192.168.9.3 dev tun7
route add 192.168.10.3 dev tun12
route add 192.168.11.3 dev tun12

route add 192.168.1.5 dev tun11
route add 192.168.2.5 dev tun11
route add 192.168.3.5 dev tun10
route add 192.168.4.5 dev tun10
route add 192.168.5.5 dev tun6
route add 192.168.7.5 dev tun11
route add 192.168.8.5 dev tun10
route add 192.168.9.5 dev tun10
route add 192.168.12.5 dev tun11

route add 192.168.1.6 dev tun8
route add 192.168.2.6 dev tun8
route add 192.168.3.6 dev tun4
route add 192.168.5.6 dev tun9
route add 192.168.6.6 dev tun9
route add 192.168.7.6 dev tun8
route add 192.168.10.6 dev tun9
route add 192.168.11.6 dev tun9
route add 192.168.12.6 dev tun8
```

3.4.2 Comparison Between LTCP and ns2

Table 3.6 below lists the comparative points of LTCP and the ns2 network simulator. Many recognized network simulators [94, 95, 112, 96] support TCP simulation functions, but, ns2 has been widely used for TCP simulation in recent research. Several points are detailed below.

Firstly, ns2 was designed as an "abstract concept" for TCP implementation. Thus, the TCP functions of dynamic window advertisement and SYN/FIN connection establishment/teardown are ignored during simulation. No real data is ever transferred during the ns2 TCP simulation. LTCP maintains all TCP functions, and can claim to output "realistic" simulation data. Secondly, ns2 was implemented as a software application, so simulation interactions with network devices and OS are hard to observe. LTCP is based on the real Linux system, so simulation interactions with OS and network devices are naturally achieved, and the user interface is simple. Specific utilities are necessary for ns2 to analyze the simulation data. but, for LTCP, all Linux based network applications/utilities can be used to analyze the simulation data, so users can reduce time to learn new utilities. Furthermore, users can choose their favorite tools during their network simulation. Even though LTCP is based on the real Linux OS, more than 256 virtual links can be built in each virtual simulated topology, which is enough for most TCP simulations.

Table 3.6: Comparisons between LTCP and the ns2 network simulator.

	LTCP	ns2
implementation of abstract TCP concept	No	Yes
observation of TCP interaction with OS	Easy	Difficult
observation of TCP interaction with network device	Easy	Difficult
combination of real network utilities and application	Yes	No
handling complicated TCP traffic	Yes	Yes
further extension	Easy	Easy

3.4.3 Comparisons Between LTCP and HTCPIP

Table 3.7 below lists the comparative points of LTCP and HTCPIP. Several points are detailed below.

LTCP references some mechanisms presented in HTCPIP, based on BSD UNIX. LTCP is constructed at RedHat 9.0 with Linux kernel 2.4.20. Linux is a UNIX-like operating system and also implements BSD Sockets as a means by which applications access TCP. The structure of the TCP/IP source code, buffer maintenance and TCP timer are totally different than those on BSD UNIX. Even though the tunnel device is supported in both operating systems, the tunnel implementation is quite different. The features of the TCP timer implementation are also different in the two systems. The upshot is that it is not a straightforward task to implement the methodology from the BSD UNIX to the Linux environment. Some challenges must be overcome, as discussed in the following sections.

Moreover, LTCP contributes "Packet Drop Generator" function to simulate the link error, which was not proposed in HTCPIP. LTCP further extends the structure connecting with the real network for the emulation objective[4], addressed in Chapter 4.

The Structures of the TCP/IP Stack

The TCP/IP stack structures are different in the two operating systems. For instance, the TCP buffer is one of the most important structures and is used to hold both incoming and outgoing data, including all layer header information, throughout all TCP/IP stacks. However, the operation of the buffer structure is different in the two systems. In Linux, it

[4] This function will be integrated in the new HTCPIP version, NCTUns network simulator.

only maintains the **sk_buff** structure for all usages of every layer. However, in BSD UNIX, there are several buffer structures, such as **mbuf**, **inpcb** and **tcpcb**, which are maintained for different usages between each layer. Because of the different operating methods, it was necessary to learn the packet transmission features in both the BSD UNIX and the Linux source code, to help trace the data flow through the Linux kernel. Compared with BSD UNIX, few books or academic papers discuss the Linux network source code systematically. This meant having to understand the Linux TCP/IP kernel by examining the source code.

The Tunnel Network Device

The tunnel network device is supported on the UNIX and Linux operating systems. However, the tunnel (TUN) implement style is quite different. For instance, Linux implements a tunnel device as a loadable module, where a new kernel function can be inserted into a running kernel without recompiling or rebooting. Implementations of a TUN device on user space programs, such as device invocation, IP addresses and routing configuration, have been redesigned as loadable Linux modules. Additionally, changes to the TUN source code were necessary to adapt the TUN device for use with our bespoke context switching interfaces.

The TCP Timer

The structure of TCP timer maintenance used in [96] is no longer in existence in current operating systems, including BSD UNIX and Linux. In [96], BSD UNIX with kernel version 2.2.8 implements TCP timer under *tcp_fastimo()* and *tcp_slowtimo()* [113]. But the TCP timer constructer was completely rewritten for BSD UNIX some years ago. The change was necessary in order to scale TCP to large numbers of active connections to improve TCP performance on fast, congested networks, by allowing finer grained retransmission timeouts. Unlike BSD UNIX, Linux TCP implies and refines many specifications from *Request For Comments* (RFC) to improve network efficiency [114]. Because their timer structures are completely different, a new methodology of timer structure has been designed to synchronize the TCP timer with the current virtual timer.

Tracing of TCP Variables

cwnd and *ssthresh* are regularly used to observe TCP behaviour during simulation. But, they are implemented as variables of TCP stack, which are invisible from the user space. The design of Linux TCP stack does not support any channel to trace *cwnd* and *ssthresh.* For this reason, LTCP implements two functions into TCP stack, which can record variations of *cwnd* and *ssthresh.* The record files can further be drawn for figures. Many graphic *cwnd* figures have been shown in Chapter 5 and will not be displayed here.

The structure comparison between LTCP and HTCPIP network simulator are summarized in Table 3.7.

Table 3.7: Structural comparison between LTCP and HTCPIP network simulator.

	LTCP	HTCPIP
operating system	*Linux*	*BSD UNIX*
TCP timer structure	*RFC definitation*	*tcp_fastimo(), tcp_slowtimo()*
tunnel device	*Linux TUN*	*BSD UNIX tunnel*
packet loss simulation	*Yes*	*No*
extended for network emulation	*Yes*	*No*
record TCP *cwnd* and *ssthresh*	*Yes*	*No*

3.5 Useful Tools for Education

LTCP can be used for teaching and understanding TCP protocol in literature rooms. TCP is one of the complex network protocols on the Internet. Students might confuse some TCP behaviours and mechanisms if they acquired their knowledge of TCP from a theory book. By means of LTCP, TCP experiments and simulations can be modelled and demonstrated in the classroom for teaching purposes. Coursework based on LTCP can also be designed, giving students significant understanding of TCP topics. Since most of the network related applications and utilities used in LTCP are exactly the same as used in Linux operating system, TCP teaching cases can be beneficial in two ways. Firstly, LTCP is “friendly” for students. Students use the same network utilities and applications in operating systems (which they already know), so LTCP simulation for students is almost the same as what they use in daily activity. Since they know how to operate the network applications and LTCP, they spend less time learning about LTCP and have more time to experiment with different scenarios.

Secondly, LTCP demonstrates how a “real” network works in reality, thus, increasing students’ study interest. LTCP can also be used for the teaching of Linux TCP/IP stack. LTCP modifies some source codes in the Linux TCP/IP stack, such as disable TCP checksum and TCP time reference, to achieve simulation purposes. Through the introduction to the LTCP source code, students can better understand about the Linux TCP/IP stack. To summarize, LTCP can be used for teaching objectives in the following subjects:

- LTCP can be used to present TCP performance while using different network conditions. For instance, LTCP can present TCP performance variation in different queuing disciplines, link delay and packet drop rate, so students can study TCP behaviour in different network topologies. Besides, LTCP can also be used for teaching how to compare TCP performance between different TCP versions by simply change TCP versions from the Linux module.

- LTCP can help the observation of TCP states. For instance, LTCP users can observe the variation of TCP slow-start algorithm, self-clocking mechanism, congestion control algorithm and fast retransmit and fast recovery algorithm. Furthermore, many TCP relative parameters, such as TCP *cwnd*, TCP *ssthresh*, duplicate ACK and RTO, can also be traced using Linux network utilities. (e.g. tcudump, tcptrace).
- LTCP can be used to teach students how to set the routing path into the Linux routing table.
- LTCP can be used to teach students how TCP performs in the wireless network and the heterogeneous network.
- LTCP can be used to teach how to generate the packet flow, how to do the queue build up and the effect of packet drops.
- LTCP can be used to teach how to develop TCP functions inside the Linux kernel.
- LTCP can be used to introduce functions of Linux TCP/IP stack.

3.6 Summary

This chapter has presented a novel TCP/IP network simulator, LTCP. LTCP features are user friendly and easily extensible to simplify TCP simulation. LTCP enables evaluation of TCP performance, just as in a complex network environment. TCP behaviour can also be traced from *cwnd* and *ssthresh* provided by LTCP. The simulation statistics are realistic, because the results are close to real network emulation. TCP Vegas was developed, which have not previously been included in the Linux kernel 2.4, to enhance the simulator functions.

Three TCP versions were further discussed in specific conditions. The results show that LTCP can certainly react to the features of different TCP versions. Through the demonstration, it is clear that Linux based network utilities can be simply used as analysis tools. Various simulated tool selections can provide users with a friendly environment without having to learn new applications. Comparisons between LTCP and ns2 have clearly shown that LTCP has many advantages in TCP simulation. The features of “friendly” and “realistic” output makes LTCP useful for TCP teaching in literature rooms. The competitive features of LTCP can help TCP research in the future.

Chapter 4

A Heterogeneous Network TCP/IP Emulator

This chapter introduces a framework for a wireless TCP emulator[1], an extension of the simulator LTCP described in Chapter 3. It can be used to monitor TCP behaviour in live wireless network, but a simple and economic framework is necessary. The emulator also enables simulation of novel protocols designed for data delivery in mixed wireless and wired networks.

In live network emulation, observing how TCP reacts in wireless links under a heterogeneous topology faces two main challenges: firstly, error sources are difficult to distinguish. In a heterogeneous network, the TCP sender and receiver transmit data crossing both wired and wireless links. When the packet loss is detected, it is hard to find the loss event caused by the wired or wireless links. Secondly, the emulation of a complex heterogeneous network is usually supported by expensive hardware devices, such as processors, network links and routers. The emulator presented in this chapter can resolve the challenges using a simple solution.

Based on the emulator framework, a complex topology of a wired network is fully simulated in only one processor, named the simulated host. Then, the simulated host is connected to a live wireless network. In this way, the network parameters of the wired network can be controlled through the simulated host. Thus, the TCP behaviours on the wireless link are the focus.

More advantages can be noted. Firstly, large numbers of computing processors are no longer necessary. Secondly, the emulator is built on top of a Linux TCP/IP stack, so it is easier for users to monitor interactions and details during the emulation. Thirdly, the wired network topology is simulated by LTCP. Thus, the user can easily separate factors when configuring wireless networks.

[1] The difference between the emulator and simulator will be discussed in the next section.

4.1 Emulator and Simulator

This section describes the difference between network simulators and emulators. Network simulators are implemented as application software running on a single computer. Network simulators take an abstract description of the network traffic and provide for conceptual virtual network environments. Network simulators are highly configurable and extensible tools designed to test and evaluate different network topologies. Network simulators usually support a virtual timescale and a controllable environment. Thus, they can be used to test new networking protocols or change currently used protocols in a controlled and reproducible environment.

Network emulators appear to be networks. End systems, such as a PC, can be built in the same way as the emulator and will behave as if they are part of the network. Network emulators can be used to observe variation in real network environments, since they are part of network end systems.

In this thesis, a "network simulators" is defined as a program designed to evaluate how the network would behave. The "network emulator" is defined as a device (computer) connecting with the real network and appearing as part of the network environment. "Emulation" is defined as network demonstration based on the emulator in the real network environment.

4.2 Emulator Features

The current emulation environment can be divided into two parts: a simulated wired network and a real wireless network as shown in Figure 4.1. The wired network is simulated by a simulated host, which runs a Linux operating system. This host acts as a "wired network box", in which wired network topologies are simulated reusing the real Linux TCP/IP stacks of the simulated host multiple times. The fixed host (FH), routers and links are simulated inside this "network box". There are two main advantages. Firstly, wired network parameters, such as link features, queueing disciplines and TCP settings are controlled through the simulated host. Secondly, a complex topology of wired networks is simulated at one processor; hence, maintenance of large processor arrays is unnecessary.

The simulated host then connects with the real wireless network instead of the simulated wireless network, because the emulator is designed to realistically construct a heterogeneous environment for TCP/IP performance evaluation and behaviour observation. Through the connection with a real wireless network, some wireless features, such as quick handover, radio jamming and dynamic data rates are naturally included. Thus, the emulator can provide a realistic reflection. If different types of wireless network are connected, the emulator also adapts for TCP performance estimation in different wireless protocols.

The emulator can work with a Linux module. For instance, the output link simulation would use different fair queueing disciplines. Most of them have been implemented as network modules in Linux. Hence, Random Early Detection (RED) or Stochastic Fairness Queueing (SFQ) can be loaded from the module *sch_red.o* or *sch_sfq.o* separately, without the need to compile a new kernel. Congestion control mechanisms, such as Explicit Congestion Notification (ECN), can be dynamically invoked through the Linux system call. In this way, the different simulated objectives are accomplished by loading the essential Linux modules.

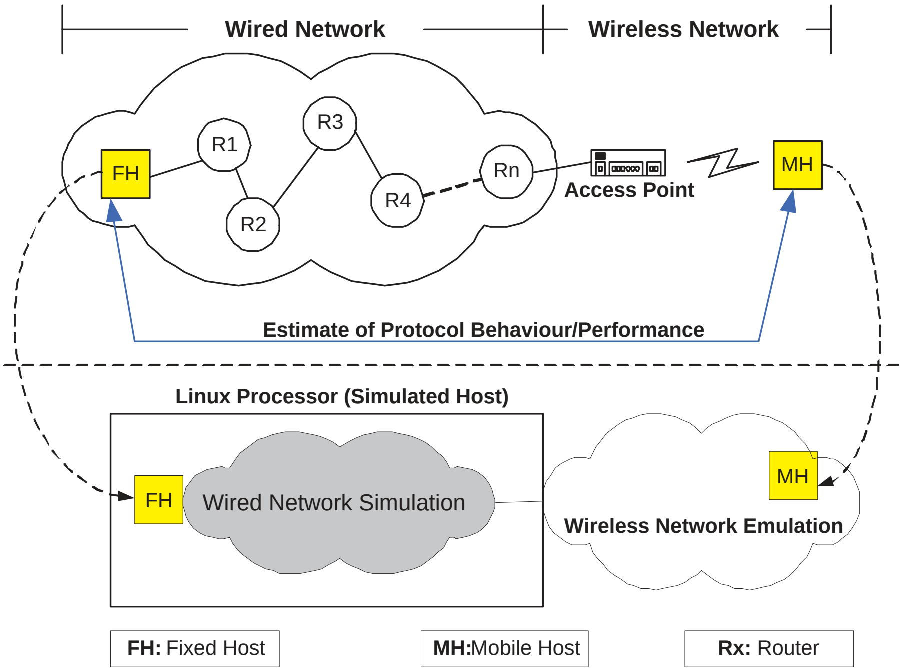


Figure 4.1: The emulator is constructed on a Linux processor. The wired network is simulated at this Linux processor acting as a "network box". The processor then connects with the real wireless network to form a "heterogeneous network".

Simulation parameters are adjusted through the standard Linux network interface. For instance, the maximum transfer unit (MTU) can be set to 512Bytes through command "ifconfig tunX mtu 512", at which tunX acts as a physical network interface on the simulator. Similarly, optional TCP algorithms, such as Delay Acknowledgement (ACK), Selective ACK, TCP timestamps and window size, are changeable through TCP parameters under the directory "/proc/sys/net/ipv4/". Two main advantages are displayed: firstly, users learn painlessly to adjust the parameters inside the emulator; secondly, existing Linux "ManPages" are naturally included as documentations of the emulator. Hence, both user learning time and emulator development time are reduced.

Linux network applications and utilities are implemented for simulation and as analysis tools. For instance, "ftp" is used to generate a simulated traffic flow and "ping" is used for round-trip time (RTT) measurements. "tcpdump" utility is helpful for traffic monitoring at each network device. The advantage of this feature is that the tools used are precisely those used in monitoring real networks.

4.3 Emulator Architecture

The emulator is based on the Linux operating system. The emulation involves five main steps: 1. packet re-routing, 2. packet arrival, 3. packet departure and 4. link simulation. 5. PDG has been discussed in Chapter 3. Figure 4.2 shows the relationships among these five parts. The following sections provide details.

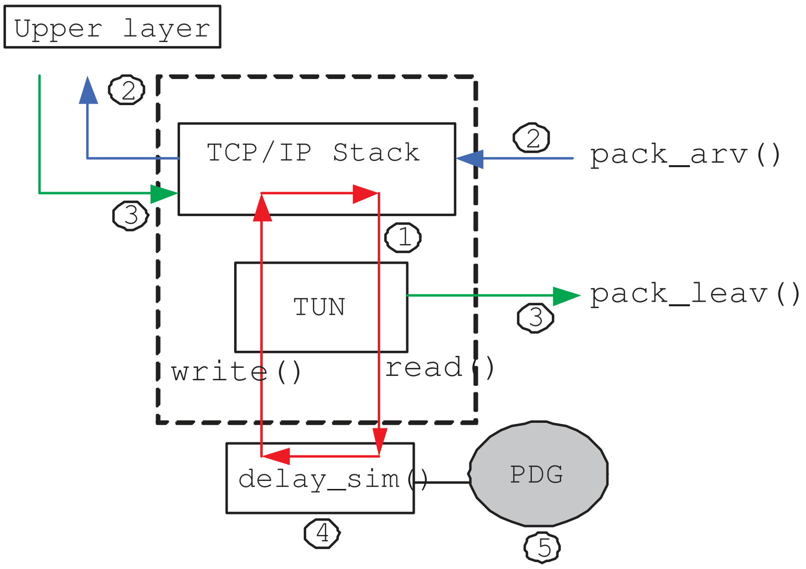


Figure 4.2: Five main parts of the emulator.

4.3.1 Packet Re-routing In and Out of the TCP/IP Stacks

The emulator forms a virtual wired network by re-routing packets in and out of the TCP/IP stacks of the Linux host multiple times. This host is LTCP enabled. The wired network is simulated on it. Surrounding the host is the real wireless network. The wired simulation and wireless network form the framework. Based on the framework, the "Private virtual IP address" mechanism is implemented to simulate the virtual nodes. The "As-seen-by-node(i)" algorithm is used to form multiple routing paths between the virtual nodes and the TUN virtual devices [106]. Each TUN device acts as an individual physical network device. Through the above mechanisms, a network topology is built up, and network traffic is re-routed between the TCP/IP stacks and TUN devices, as displayed in Figure 4.2 (label 1). Therefore, a complex wired network is achieved by only one PC. Since the real network kernel is implemented to fake a complex wired network, a “realistic” wired network environment is created.

The concept of using virtual nodes and virtual devices inside the kernel for network simulation is not novel, and several efforts have been made [96, 97, 21]. However, the present framework differs from other projects, because :

- [96] proposed its mechanisms built on top of FreeBSD, but the simulator I propose is built on top of Linux. In addition, the simulator can build a framework, which can then become an emulator.
- The framework we use here combines a real wireless network. It is enhanced by adding two additional functions and will later be described in Section 4.3.2 and 4.3.3.
- The emulator has full cooperation with Linux networking modules, so the function extensions are available.

4.3.2 Packet Arrives at the Simulated Host

The function pkt_arv() deals with the emulated traffic incoming from the real network. It starts when the packet is received by the Linux Network Address Translation (NAT) module and ends when the packet is received by the applications. The flowchart summarizing the algorithm for packet arrival in Figure 4.3 is described below.

1. Simulation non-executing state: If the emulator is not executed on the simulated host, the packet is processed as normal.
2. Simulation executing state: If the simulation is executed, the nat_trig() function is called to invoke the Linux NAT module, and delivers the “Private virtual IP address” information to the NAT module. Then, the function of the NAT *prerouting chain* is triggered to rewrite the packet’s IP address as specific “Private virtual IP address”. In this way, incoming packets from a real network can be connected with the “As-seen-by-node(i)” routing mechanism, and go to the “route state” as described below. It is important to mention here that the emulator reuses any of the existing Linux functions instead of rewriting them, so that development time is reduced and any additional Linux functionalities can be used.
3. Route state: When the packet goes into the route state, the packet starts to re-route between TCP/IP stacks and TUN devices, as mentioned in Section 4.3.1. Each time a packet arrives at the TCP/IP stacks, the packet’s next routing path is referenced from the “As-seen-by-node(i)” mechanism setting in the kernel routing table. If the next node is a virtual host, the pkt_arv() function leaves the route state directly and goes into the host state. If the next node is a virtual router, the packet re-routes again.

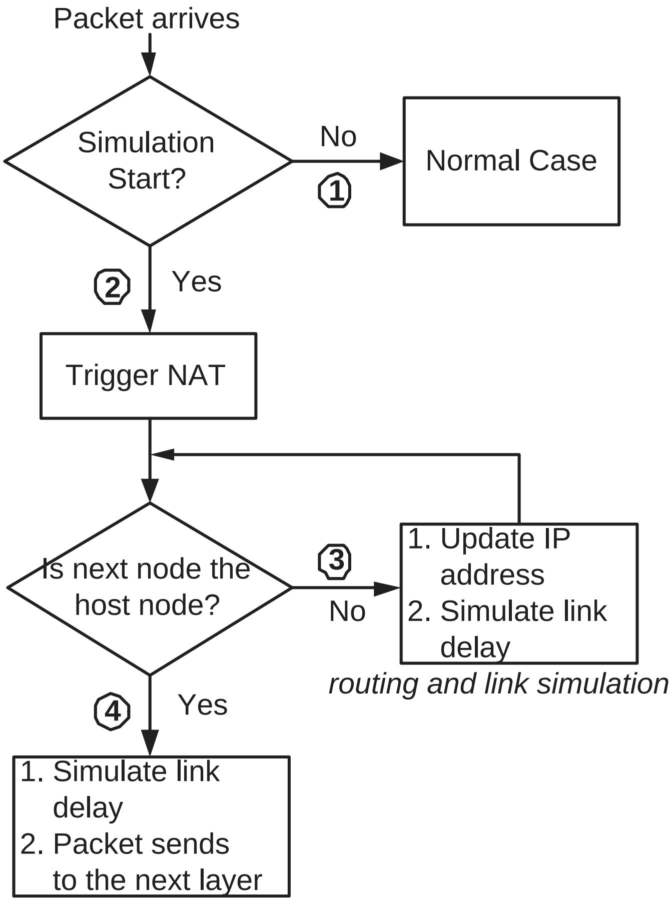


Figure 4.3: The flow chart for a packet arriving at the simulated host.

Hence, if 10 virtual routers are set in the network topology, route state will be executed 10 times until the packet arrives at a host. Each time the packet arrives at the TUN device, the link_sim() function is invoked to deal with the link relative simulation. A detailed description of the link_sim() function is shown in Section 4.3.4.

4. Host state: The packet is prepared to pass to the upper layer (i.e. TCP, UDP or ICMP).

4.3.3 Packet Leaves the Simulated Host

The pkt_leav() function deals with the emulated traffic outgoing from the application layer. It starts when the packet is received by the Linux TCP/IP stacks and finishes when the packet is received by the Linux NAT module. A flowchart summarizing the algorithms for pkt_leav() is provided in Figure 4.4 below.

1. Route state: It has similar functionalities with the pkt_arv() function, referred to earlier.

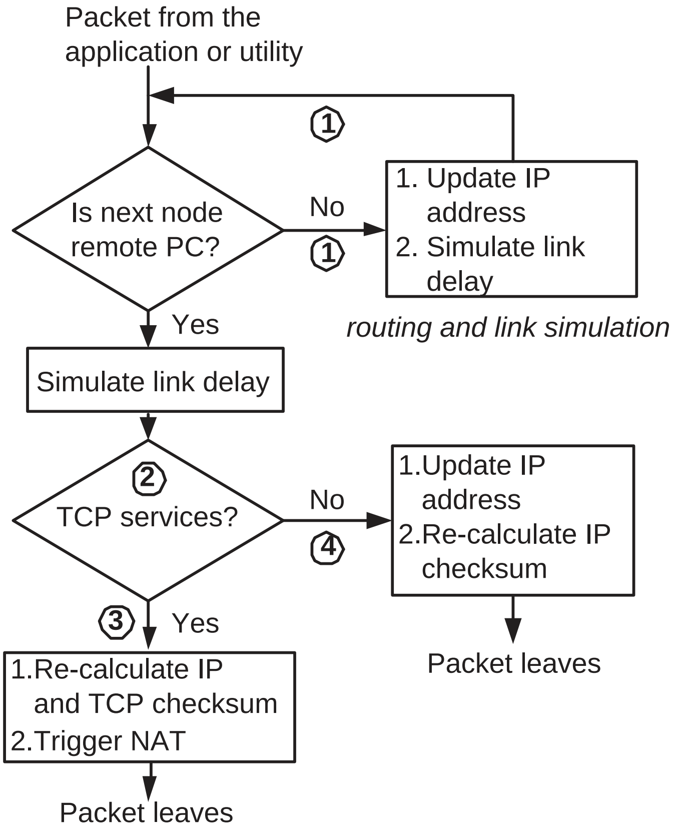


Figure 4.4: The flow chart for a packet leaves the simulated host.

2. Judge state: When the packet leaves the route state, it means that the next node is the remote PC. Hence, additional tasks must be carried out before the packet is injected into the real network. A main task here is to differentiate varieties of service type. For instance, if the network service type belongs to TCP, the packet is delivered to the TCP state for additional processes. If the service type belongs to other protocols, the packet is delivered to the relative state for additional processes. Here, the Internet Control Message Protocol (ICMP) service is used as an example.

3. TCP state: When the packets go into the TCP state, the IP and TCP checksum of the packet is first re-calculated. Otherwise, the packet will be dropped by the router or remote PC due to checksum error. Then, the nat_trig() function is implemented again to invoke the Linux NAT module and asks for IP address redirection. nat_trig() delivers the remote IP information into the NAT module, in which the function of NAT *postrouting chain* is triggered to change the packet's IP address from "Private virtual IP address" to the remote PC's address. After leaving the NAT module, the packet is injected into the real network.

4. ICMP state: In this state, the packet's IP address is first updated from "Private virtual IP address" to the remote PC's address. Then, the IP checksum of the packet

is recalculated. After this, the packet is injected into the real network. Two points are worth to mention: firstly, the Linux NAT module is not implemented here because the function NAT *postrouting chain* is not allowed to serve ICMP packets; secondly, since the packet belonging to different protocols is separated, it is unnecessary to re-calculate the TCP checksum at the time. Hence, emulator overload is reduced, especially in long term simulations.

4.3.4 Link Simulation

The link parameters, of which there are many types, such as link delay and link bandwidth, are simulated in the link_sim() function, which is triggered when packets arrive at the TUN device. After packets arrive at each TUN device, the user space program invokes the *read()* system call to gather packets from the TUN device and then store them in user buffers. The simulation of link bandwidth and link delay are executed. When the link simulation is done, the *write()* system call is processed to pick up packets from the user buffers and then restore packets to the TUN device. The link simulation is accomplished between each *read()* and *write()* event. The relationship between each *read()* and *write()* event in link simulation is shown in Figure 4.5

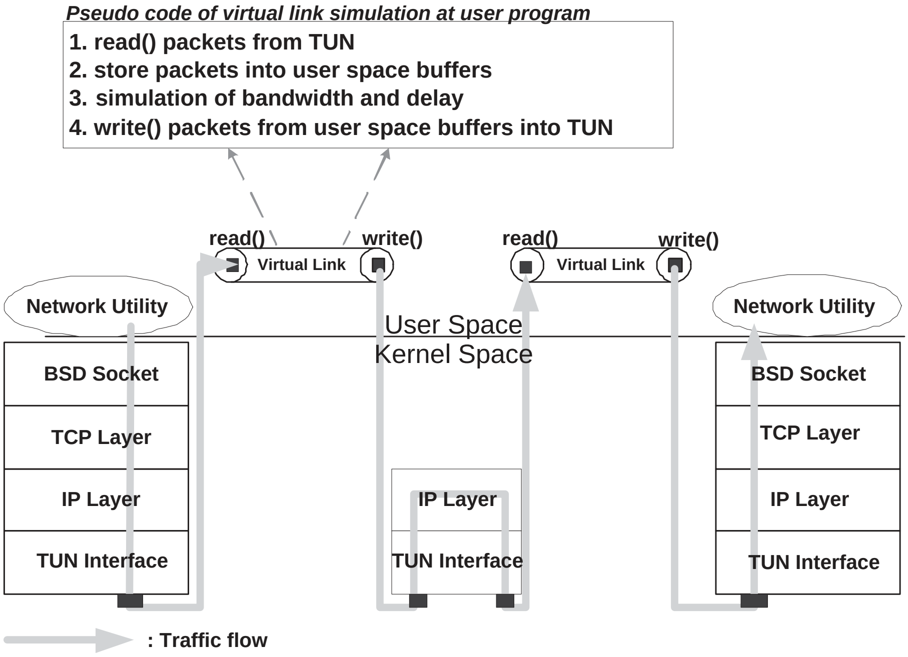


Figure 4.5: A packet stored time between *read()* and *write()* system call is used to make the link simulation.

4.4 Demonstration of TCP in a Real Wireless Network

In this framework, we try to form a heterogeneous network, which is composed of a wired network and a wireless network. The wired network is simulated by the simulator on a Linux host. The wireless network is real. When controlling the wired network parameters, it is presumed that no packet loss occurs via the wired link. Thus, the observation can focus on the packet loss in the wireless link. The next sections present the emulator results of three different TCP versions.

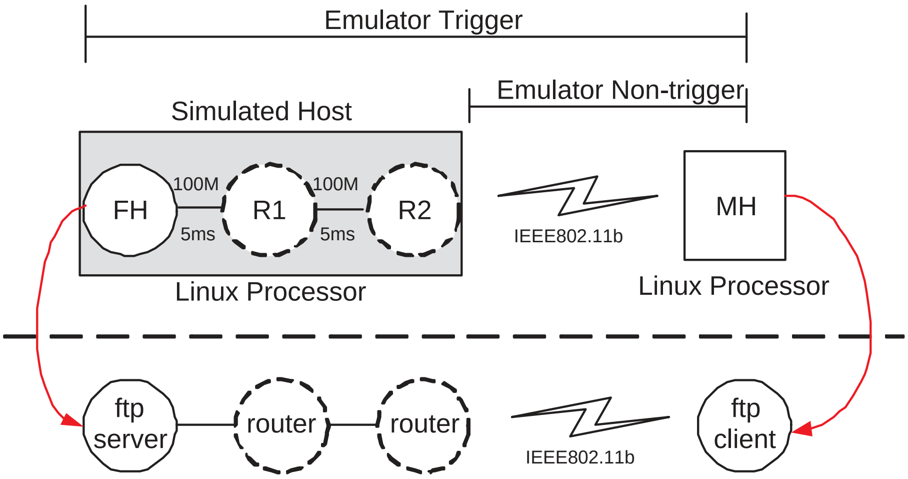


Figure 4.6: Network topology for experiences.

The emulator was constructed on two Redhat Linux processors using kernel version 2.4.20, to be precise. Figure 4.6 shows the network topology. One Linux processor is constructed as a simulated host (wired network) and another Linux processor acts as a mobile host. Three virtual nodes (one host, two routers) and links are set up inside the simulated host. FH is implemented as a source node, in which an ftp server is set up. R1 and R2 are routers. The simulated host then connects a wireless router (ASUS WL-500g wireless router), which is used as a base station (BS). MH is a mobile host, which is set up on a laptop. A Cisco aironet 340 PCI wireless card is integrated on the MH to connect with the BS. The bandwidth between FH and R1 is 100Mbps, with 5 milliseconds (ms) propagation time. There is no bottleneck link during any wired nodes. The buffer size is set to maximum and no extra packet loss situation is set on the wired network link. The MSS is set to 1000 bytes when the packet is sent out from the FH. The wireless network crossing R2 and MH is IEEE 802.11b [6]. It is worth noting here that the packet will be segmented into smaller segments when the packet goes through the wireless network. This is a feature of the current wireless network and we cannot control this factor during the demonstration. Based on this topology, it is clear that we can fully control the wired network part without any packet loss event, so features of the wireless network are the main influence on TCP behaviours. The experiment results follow.

4.4.1 Emulation Methodology

The measured results in different time slots within one day are shown in Table 4.1. There are six time slots, each slot for one hour, within the day. For each time slot, nine tests were conducted, three for NewReno, three for SACK and three for Vegas, in the following sequence (each test lasting five minutes): A NewReno connection was first tested for five minutes and after its completion, a test for SACK was started one minute later for five minutes, followed by the test for Vegas and so on. For each time slot, the distance between MH and BS was also changed. Thus, we can use the correlation between packet loss and distance (strength of signal) to observe the TCP behaviours over the wireless link.

The tests for the six time slots were conducted over five days (Monday to Friday). The general trends over five days were similar, so only the results from one day are shown in Table 4.1. The result for each time slot represents the test average for the different TCP versions. Detailed results for time slots are presented from 9:00∼10:00 in Table 4.2.

Table 4.1: The TCP emulation for the six time slots in one day. (Distance: the distance between BS and MH. Th.: average throughput, Rtmit. Pkt.: retransmit packets, To. Trs. Pkts: total transmit packets, Pkt. Err. Rate: (Rtmit. Pkt. / T. Tst. Pkt.))

	NewReno / SACK / Vegas		
Time slots	9:00∼10:00	10:30∼11:30	12:00∼13:00
Distance(m)	10	15	20
Th.(KB/s)	214.4/210.2/203	197/194.3/185.7	171.7/172.3/162
Rtmit. Pkt.	14.3/14.7/14.3	43/33/34	94/100.3/99
T. Tst. Pkt.	21479.7/21075.3/20331.3	19744/19504/18620	17252.3/17520.1/16276.6
Err. Rate (%)	0.067 /0.07 / 0.07	0.218 /0.169 /0.183	0.545 /0.572 / 0.608
Time slots	13:30∼14:30	15:00∼16:00	16:30∼17:30
Distance(m)	5	1	very close BS
Th.(KB/s)	233.3/232.7/225	251/251.3/245.7	251.3/251.3/244
Rtmit. Pkt.	2/3/5.2	0.3/0.3/0	0/0/0
T. Tst. Pkt.	23350.3/22213.7/22577	25131.3/25145.3/24900	25133/25145/24820
Err. Rate (%)	0.009 /0.014 /0.023	0.001 /0.001 / 0	0 / 0 / 0

4.4.2 Results

Table 4.1 shows that wireless transmission stability is in inverse proportion to distance, due to the decrease in signal. When the MH is very close to the the BS (16:30∼17:30), the error rate is close to 0%, but when the distance between the BS and the MH is increased, the error rate increases to 0.5% (12:00∼13:00). The error rate is equal to the *Retransmit Packets / Total Transmit Packets*. Total Transmit Packets are the number of transmit packets for each 5 minutes' emulation. Retransmit Packets are equal to the *loss packets + bit corrupt packets*. It is worth noting here that numbers of new wireless protocols have implemented

Table 4.2: The TCP emulation in only one time slot.(Th.: average throughput, Rtmit. Pkt.: retransmit packets, To. Trs. Pkts: total transmit packets)

9:00∼10:00				
NewReno	Th(KB/s)	214.2	215.3	213.8
	Rtmit. Pkt.	14	10	19
	To. Trs. Pkts	21456	21564	21419
SACK	Th(KB/s)	215	201.1	214.6
	Rtmit. Pkt.	10	15	19
	To. Trs. Pkts	21593	20142	21491
Vegas	Th(KB/s)	202	203.4	203.5
	Rtmit. Pkt.	15	12	16
	To. Trs. Pkts	20278	20351	20365

novel technologies into their MAC layer to quickly recovering some lost packets between the BH and the MH. Over these wireless protocols, TCP sender site will not be aware of some lost events, so TCP performance would not be affected by those lost packets. Moreover, some slight bit corruptions in packets can also be fixed, so retransmission is unnecessary. The considered Retransmit Packets are the actual retransmitted packets traced from the TCP sender. Compared with some simulation tools, such as ns2 and OMNeT++, the TCP checksum protocol and the wireless recovery mechanism have not yet been considered, Our emulation result is competitive and can realistic react with TCP over the wireless link.

NewReno and SACK was always observed to perform better than Vegas. This is because the main idea of Vegas is to “prevent” packet loss in buffer overflow, but, in this network topology, the loss event comes from random wireless loss, which Vegas cannot take advantage from.

It was also found that NewReno and SACK performance was almost the same in the six time slots. This result is quite different from those presented in Section 3.3.4, where TCP was simulated in the wired network topology with limited queue sizes. This is because SACK makes an improvement on NewReno to prevent retransmission timeout due to multiple packet loss in one TCP window. However, this emulation showed random packet loss occurring separately over the wireless link, so NewReno only triggers its fast retransmit and fast recovery algorithm to retransmit the lost packet without falling into the timeout event.

4.4.3 Summary of Wireless TCP

With reference to wireless TCP emulation, two steps are proposed to improve TCP performance in a situation of random packet loss. Firstly, TCP must be aware of loss types. Secondly, adjustment of TCP behaviours should occur when the loss event is separated. If the event occurs in congestive loss, TCP keeps the behaviour as normal. But, when random loss is detected, TCP could still keep a high *cwnd* without slowing down the transmission speed, so the TCP throughput can be improved. From the emulation result discussed in this chapter, we find that the retransmission timeout is not the main reason that slows down TCP in the wireless links, so we do not focus on the timeout situation in our mechanism. The mechanism will be discussed in the next chapter.

4.5 Summary

This chapter presents a TCP/IP emulator. It is based on the simulator presented in Chapter 3. In order to model the real wireless environment, we build a framework, which contains one or more mobile host(s) and a wireless access point. The wired network is simulated by the simulator, LTCP, but the wireless network is real. Therefore, what happen in the wireless network can be clarified. In order to verify that the emulator functions well, three TCP versions have been used for testing. In testing, it was found that packet loss may cause throughput reduction, but TCP should react differently if the reason for packet loss can be detected. Thus, the next chapter will describe efforts to distinguish packet loss types.

Chapter 5

TCP NewZag Mechanisms

This chapter proposes a novel end-to-end congestion control mechanism called TCP NewZag. NewZag is simple and effective in dealing with wireless random packet loss, so it can improve TCP performance in the heterogeneous network. The key concept of NewZag is to monitor the network round trip time and amounts of packet loss. The feedback information is then used to decide whether the packet losses are likely to be caused by random packet loss or congestion loss. Specifically, TCP NewZag proposes two new mechanisms: (1) a new end-to-end loss differentiation algorithm (LDA); (2) adjustments to the TCP NewReno multiplicative decrease algorithm. These two mechanisms require adjustment only to the TCP sender side, while the receiver side protocol remains the same. This feature is important for further development of NewZag because the effect upon the real environment can be minimized. More details about these mechanisms are provided later.

Experimental measurements on a single link show that NewZag achieves significant throughput improvement when wireless random packet losses occur. Compared with NewReno with a 0.5% random packet loss rate, 20% throughput improvement can be demonstrated. The throughput of NewZag is higher than that of selected versions of wireless TCP in different random loss probabilities. Experiments on a competitive link prove the fairness of NewZag. It provides better performance even when it shares a common link with other wireless TCP mechanisms.

This chapter detailing the NewZag mechanisms starts with a discussion of the relevant literature. Then it introduces the loss differentiation algorithm followed by a discussion of the modification of the TCP sender when random packet loss is recognized. Next, the loss differentiation algorithm is verified a performance evaluation of NewZag is made. The chapter concludes with the experimental results comparing it with other well known wireless TCP mechanisms in live network experiments.

5.1 Relevant Literature

The development of TCP NewZag refers to four proposed TCP schemes: Biaz Scheme, Spike Scheme, ZigZag Scheme and TCP Veno.

- Biaz Scheme[81]: The Biaz scheme uses packet inter-arrival time to distinguish congestion packet loss from random packet loss. Figure 5.1 shows how the algorithm works. Equation 5.1 presents the main idea,

$$(n+1)T_{min} \leq T_i < (n+2)T_{min} \tag{5.1}$$

  T_{min} is the minimum packet inter-arrival time that the receiver has measured so far. n is the number of packet losses. Precisely, P_i denotes the last in-sequence packet received before a loss event, and P_{i+n+1} is the first received out-of-order packet after a loss occurs. By definition of a TCP connection, there are n possible lost packets between P_i and P_{i+n+1} and T_i is the interval time between P_{i+n+1} and P_i. If all packets are the same size, $(n+1)T_{min}$ should be the minimal arrival time that P_{i+n+1} can make. $(n+2)T_{min}$ is the acceptable delay time. Thus, the relation between T_i and T_{min} is used to differentiate two loss types. If $(n+1)T_{min} \leq T_i < (n+2)T_{min}$, the lost packets are assumed to be the result of random packet loss. Otherwise, congestion loss is assumed.

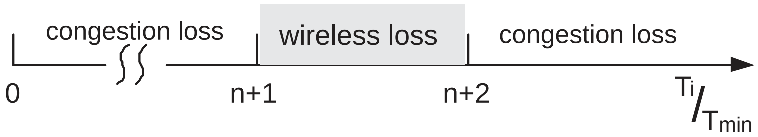


Figure 5.1: Biaz scheme. n is the number of consecutive packet(s) lost. T_i is the instantaneous packet interarrival time of the first packet received after the loss. T_{min} is the minimum packet interarrival time observed so far.

  The main concept in the Biaz scheme comes from the arrival time of P_i. Three relations between P_i and P_{i+n+1} are discussed: firstly, if P_{i+n+1} does not show up at approximately the expected time, increased queuing time at the buffers is assumed. Secondly, if the P_{i+n+1} arrives before its expected arrival time, some packets are dropped at the buffer that is expected. Biaz classifies these two situations as congestion loss. Thirdly, if P_{i+n+1} arrives at approximately the expected time there is high possibility that the lost packets just occurred randomly.

- mBiaz Scheme[82]: The mBiaz scheme is a variant of the Biaz scheme. Figure 5.2 shows how the algorithm works. It involves a slight change to Equation 5.1 in the upper boundary. It assumes that if $(n+1)T_{min} \leq T_i < (n+1.25)T_{min}$ is satisfied, then these n packets are assumed to be the result of random packet loss. Otherwise, these losses are classified as congestion losses. mBiaz is used to correct over-estimate problems when the Biaz scheme is used to connect with links with high congestion losses. According to [82], when the wireless hop is the last node of the connection, the congestion loss classified by Biaz is twice as large as the real loss. This is because Biaz over-estimates the threshold of queuing congestion loss. mBiaz adjusts the thresholds of Biaz to reduce the event of congestion loss suggested by Biaz. A study of mBiaz shows that it provides good benefits of low congestion loss and high throughput when the wireless node is the last connection host.

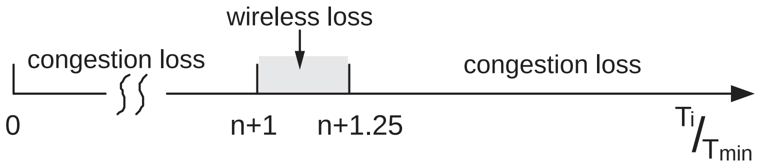


Figure 5.2: mBiaz scheme: modified Biaz scheme.

- Spike Scheme [83]: Spike is a scheme based on relative one-way trip times (ROTT) measurement. ROTT is the packet travel time from sender to receiver and Spike uses ROTT to identify the state of the connection. Figure 5.3 shows how the Spike scheme works.

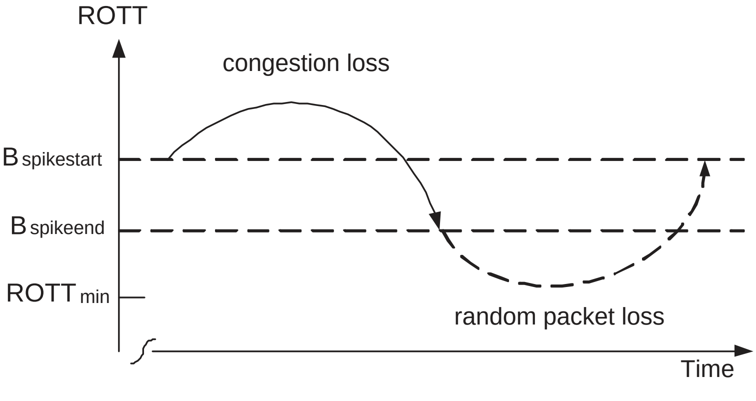


Figure 5.3: Spike scheme.

If the connection is in the *spike state*, a packet loss is classified as congestion loss. Otherwise, the loss is classified as random packet loss. Let i denote the sequence number of packet P_i and $ROTT_i$ be P_i's travel time from the sender to receiver. If the connection is currently not in the *spike state* and $ROTT_i > B_{spikestart}$, the Spike scheme enters the *spike state*. Otherwise, if the connection is currently in the *spike state*, and $ROTT_i < B_{spikeend}$, the algorithm leaves the s*pike state*. $B_{spikestart}$ is a threshold that denotes the maximum ROTT before entering the *spike state*, defined as $(ROTT_{min} + 20ms)$. $B_{spikeend}$ is the minimum ROTT before leaving the *spike state*, defined as $(ROTT_{min} + 5ms)$. $ROTT_{min}$ is the minimum measured one-way trip time. The main drawback of Spike is that fixed thresholds of $B_{spikestart}$ and $B_{spikeend}$ will restrict algorithm usage in the future, because variable link types are present in the current network. It means that a connection cannot usually experience a delay of 5 ms to 20 ms. Neither can wireless loss occurring in the situation of high ROTT be considered [84].

- The ZigZag Scheme [82]: ZigZag uses a similar notation as in the Biaz scheme. ZigZag uses the number of losses and the mean/deviation of ROTT to classify the loss type. If i is the sequence number of packet P_i and $ROTT_i$ is the travelling time from the sender to the receiver, a loss is treated as random packet loss if:

$$\begin{array}{rl} & (\mathrm{n}=1) \;\; \&\& \;\; (ROTT_i < ROTT_{mean} - ROTT_{dev}) \\ \mathrm{OR} & [(\mathrm{n}=2) \;\; \&\& \;\; (ROTT_i < ROTT_{mean} - ROTT_{dev}/2)] \\ \mathrm{OR} & [(\mathrm{n}=3) \;\; \&\& \;\; (ROTT_i < ROTT_{mean})] \\ \mathrm{OR} & [(\mathrm{n}>3) \;\; \&\& \;\; (ROTT_i < ROTT_{mean} + ROTT_{dev}/2)] \end{array}$$

Otherwise, the loss is treated as congestion loss. $ROTT_{mean}$ and $ROTT_{dev}$ are calculated using the exponential average with $\alpha = 1/32$, as follows:

$$\begin{aligned} ROTT_{mean} &= (1-\alpha) * ROTT_{mean} + \alpha * ROTT_i \\ ROTT_{dev} &= (1-2\alpha) * ROTT_{dev} + 2\alpha * |ROTT_i - ROTT_{mean}| \end{aligned} \tag{5.2}$$

According to [82], from the definition of ZigZag, ROTT has a high probability of having a value greater than $(ROTT_{mean} - ROTT_{dev})$ if it is a normalized Gaussian distributed random variable. If congestion loss goes along with higher link delay and one packet loss is the most common loss situation in a wired network, the threshold of $ROTT > ROTT_{mean} - ROTT_{dev}$ would classify most of the congestion loss. Increasing the threshold with the number of losses encountered means that high loss is associated with serious link congestion and with higher ROTT. Thus, four or more loss events would be classified as congestion loss only when a relatively large ROTT is observed. But, the ZigZag scheme cannot perform well in some network topologies [84].

- TCP Veno [85]: The LDA of TCP Veno uses the estimate of the *backlog* value as a signal to differentiate congested losses from random packet losses. The *backlog* value was first developed by TCP Vegas, as discussed in Section 2.4. If the *backlog* value is below the threshold, the loss is considered as random packet error. Otherwise, the loss is regarded as congested error. If the loss is considered congested, Veno maintains its congestion control mechanism as standard Reno. If the loss is due to a random error, Veno refines additive increase and the multiplicative decrease algorithm of Reno. The advantage of Veno is that it can efficiently use the available bandwidth in the current link. Moreover, it is compatible and can co-exist with current TCP in different network topologies. In addition, the implementation of Veno is simple on the sender side.

5.2 TCP NewZag Mechanisms

NewZag mechanisms and implementations are described separately in the next four sections.

5.2.1 The New Loss Differentiation Algorithm

The NewZag loss differentiation algorithm (LDA) is designed to recognize types of packet losses. The loss type is classified as random packet loss if:

$$\begin{array}{rl} & (\text{n} = 1) \;\; \&\& \;\; (RTT_i < RTT_{mean} - RTT_{dev}) \\ \text{OR} & [(\text{n} = 2) \;\; \&\& \;\; (RTT_i < RTT_{mean} - RTT_{dev}/2)] \\ \text{OR} & [(\text{n} = 3) \;\; \&\& \;\; (RTT_i < RTT_{mean})] \\ \text{OR} & [(\text{n} > 3) \;\; \&\& \;\; (RTT_i < RTT_{mean} + RTT_{dev}/2)] \end{array}$$

Otherwise, the loss is treated as a congestion packet loss. n is the number of lost packets, as described in detail in Section 5.2.3. The RTT_i is the round trip time of packet i. The RTT_{mean} and RTT_{dev} are calculated using the exponential weighted moving average (EWMA) , as follows:

$$RTT_{mean} = (1 - \alpha) * RTT_{mean} + \alpha * RTT_i \tag{5.3}$$

$$RTT_{dev} = (1 - 2\alpha) * RTT_{dev} + 2\alpha * |RTT_i - RTT_{mean}| \tag{5.4}$$

In these two equations, $(1-\alpha)$ is the exponential decay factor that controls the smoothness of RTT_{mean} and RTT_{dev}. With reference to [51], we define α as presenting the type 2^Y, where Y might be any integer between -8 and -2. Emulation results show that when $Y = -4$ the experiments achieve better results.

Detailing the LDA scheme, by definition from Equation 5.3 and 5.2, RTT has a high probability (around 85% greater than $RTT_{mean} - RTT_{dev}$) if it is a normalized Gaussian distributed random variable. The congestive loss in the wired links could be classified by the threshold $RTT > RTT_{mean} - RTT_{dev}$ because most of the packet losses in the wired link are caused by high queuing delay and then follows the congestive queue drop. On the other hand, if there is a packet loss and $RTT < RTT_{mean} - RTT_{dev}$, there is a high probability that this packet loss is happening in the wireless links. Thus, the loss type is classified as random packet loss in the wireless links.

The NewZag LDA described above references theoretical foundations proposed in [81, 82, 83], but has three main improvements. The first improvement is achieved by using RTT in the NewZag LDA instead of ROTT suggested by [83]. RTT is a better solution here because of two main reasons. The first reason is the time synchronization problems. ROTT is a one-way time of packet forward delay between the sender and receiver. It is calculated by taking the between the receiver's clock and the sender's timestamp, defined in Equation 5.5:

$$ROTT = RECEIVER_{accepting\_packet\_time} - SENDER_{sending\_packet\_time} \tag{5.5}$$

Since $RECEIVER_{accepting\_packet\_time}$ is the time record when the packet is received by the receiver host, so the time criterion is based on the receiver clock. But, the $SENDER_{sending\_packet\_time}$ is the time record when the packet is sending out from the sender host, so the time criterion is based on the sender clock. If there is a global time synchronization between the sender and receiver, it is an easy task to estimate ROTT. However, ROTT cannot be measured accurately, because the clocks at the end hosts are not synchronized with each other. Even though several proposals are suggested [115, 116, 117], the clock synchronization problem is not yet suitable for measuring the ROTT [118, 119]. Besides, it remains still a challenge to guarantee a synchronized clock on massive networks, such as the Internet. In summary, using ROTT as the reference time could affect the NewZag LDA and decrease the judgement accuracy of error type. Thus, RTT is used in NewZag LDA to replace ROTT, so the time clock is always referenced to the sender time. The two problems described above can be naturally solved. The NewZag LDA can avoid time reference problems, so the LDA may have higher accuracy.

The usage of a dynamic threshold is the second improvement of NewZag. In [81], two thresholds, $B_{spikestart}$ and $B_{spikeend}$, are used as time boundaries to judge packet losses coming from the wired link or the wireless link. $B_{spikestart}$ is defined as ($ROTT_{min}$ + $20ms$) and $B_{spikeend}$ is defined as ($ROTT_{min}$ + $5ms$), separately. The main drawback of the LDA described in [81] is that it is not suitable for use in topologies in which the link connection experiences extra delays lower than 5 ms or higher than 20 ms, because these two thresholds have been fixed. Thus, the NewZag LDA uses the packet loss number (n) and RTT variation as a reference number to build a dynamic threshold model, so the implementation and scalability of NewZag LDA are extended.

TCP NewReno adjustments are the third improvement of the NewZag. [82] proposes a LDA to separate the loss types which come from the wired link, or the wireless link, but it does not further discuss TCP behaviour when the loss type has been separated. Thus, NewZag proposes one TCP adjustments following the LDA step. The mechanism is discussed in the next sections.

In summary, RTT is used in NewZag LDA, so time inconsistency problems are solved. A dynamic threshold is proposed, so the implementation of NewZag LDA is extended. Moreover, two TCP adjustments are implemented after the LDA, so the TCP performance can be increased.

5.2.2 A Modified TCP NewReno

The TCP NewReno algorithm is modified when loss types are distinguished by the NewZag LDA. If the loss is considered to be congestive, NewZag maintains its congestion control mechanism as standard NewReno. If the loss is caused by a random packet error, NewZag increases the TCP *cwnd* in new strategies. The paragraphs below describe how NewZag functions when loss type is determined.

1) Slow-start Algorithm: NewZag employs the same slow-start algorithm as NewReno's without modifications. NewReno's slow-start algorithm sets the window size, *cwnd*, to one and sends out the first packet at the start-up phase of a connection. The slow-start threshold (*ssthresh*) is implemented as a *cwnd* threshold. Each time a packet is acknowledged, and the window size is increased by one, *cwnd* increases from one to two after the first round-trip time, from two to four after the second round-trip time and so on. This results in an exponential increase of the sending rate over time until packet loss is detected or *cwnd* is equal to *ssthresh*.

2) The Additive Increase Algorithm: NewZag employs the additive increase algorithm from NewReno's. NewReno triggers the additive increase algorithm when *cwnd* is larger than *ssthresh*. During this phases, the increase rate of *cwnd* is reduced to avoid congestion, that is, NewReno *cwnd* is increased by one after each round-trip time, rather than after the reception of each ACK. Essentially, *cwnd* is set to *cwnd + 1/cwnd* after each ACK to achieve a linear increase effect. In NewReno, the initial value of *ssthresh* is 64KB. The value of *ssthresh* is changed dynamically throughout the duration of the connection.

```
If ($cwnd \geq ssthresh$)
    when each new ACK is received;
    set $cwnd = cwnd + 1/cwnd$
```

Even though NewZag keeps the same additive increase algorithm, the behaviour of TCP is totally different from NewReno in this step. This is because the *cwnd* variation depends on the *ssthresh*. NewZag will modify the "multiplicative decrease algorithm" of NewReno, which affects the value of *ssthresh*. This is described below.

3) The Multiplicative Decrease Algorithm: Two schemes are used to detect packet loss in NewReno. A packet can be declared lost if it has not been acknowledged by the receiver when a timer expires. In this case, the slow-start algorithm is initialled again, with *ssthresh* set to *cwnd/2* and *cwnd* remaining at two. This has the drastic effect of suddenly reducing the sending rate by a large amount. That is, the expiration of the timer is interpreted as an indication of server congestion.

NewReno also employs fast retransmit to detect packet loss. When each out-of-order packet reaches the receiver, NewReno retransmits the last ACK to the sender. In NewReno, when the sender receives three duplicate ACKs, it declares the corresponding packet to be lost, even if the timer has not expired. A complementary algorithm to fast retransmit, called fast recovery, is then employed to alleviate the congestion as follows:

```
(1) Retransmit the missing packet
    set ssthresh = Max(cwnd/2, 2*MSS);
    set cwnd = ssthresh + 3*MSS;
(2) Each time another duplicate ACK arrives
    set cwnd = cwnd + 1*MSS;
(3) When the next ACK acknowledging new data arrives, two steps are discussed:
    (i) the ACK acknowledges all of the data up to and including "recover"
        set cwnd = ssthresh
    (ii) the ACK does "not" acknowledge all of the data up to and including "recover"
        retransmit the first unacknowledged segment
        set cwnd = the amount of new data acknowledged
```

However, from step(1), rule(1), once the packet loss is detected, the throughput can be dramatically reduced. Therefore, since the reason for packet loss is understandable in NewZag, a different method is used to set the new *ssthresh*. More precisely,

```
If (the error is due to random pakcet loss)
    set ssthresh = cwnd*(4/5)
else if (the error is due to congestion loss)
    set ssthresh = cwnd / 2
```

The main object of NewZag is to recover *cwnd* at high value when all lost packets are eliminated. If random packet loss is detected, NewZag resets *ssthresh* to $cwnd * 4/5$, so *cwnd* can be reset to a high value in step 3 (i). In general, any factor larger than 1/2 is preferred.

For illustration, Figure 5.4 and Figure 5.5 show the typical window evolution of NewZag and NewReno. For observation purposes, the setting of TCP parameters inside the Linux have been doubled. More detailed results will be provided in the next section. In this

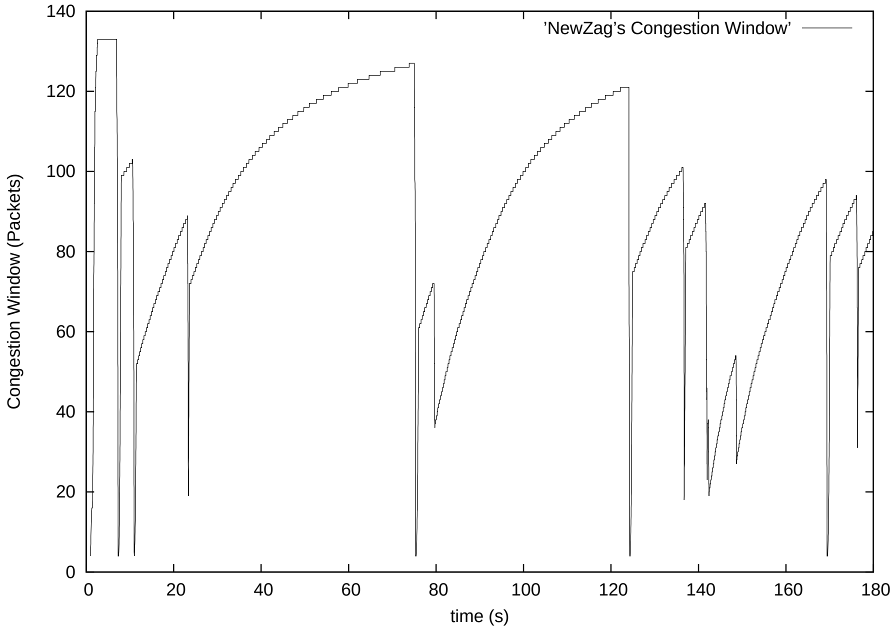


Figure 5.4: NewZag's window evolution.

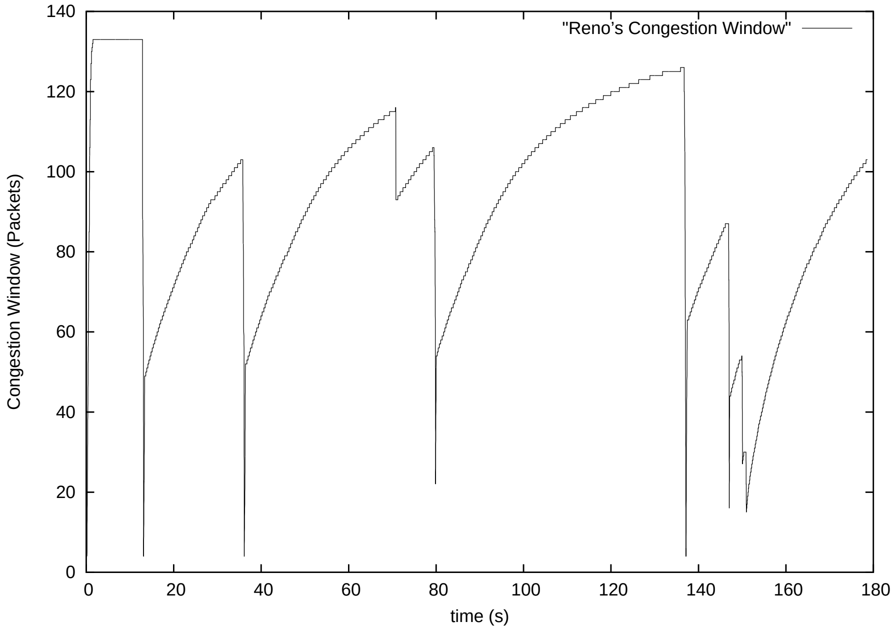


Figure 5.5: NewReno's window evolution.

example, the average of the RTTs traced by "ping" (100 packet samples) is 0.2ms in the wired links and 4.25ms in the wireless links. In the NewReno simulation (Figure 5.5), loss events happen 6 times, at 14s, 36s, 79s, 137s, 147s and 150s. At each loss event, NewReno decreases and resets *ssthresh* to *cwnd/2* and renews *cwnd* to the applicable value. However, in the NewZag simulation , packet losses happen 11 times in the network at around 7s, 10s, 23s, 75s, 79s, 124s, 136s, 141s, 148s, 169s and 176s. In this case, five error events are categorized by NewZag as congestive losses at 10s, 75s, 79s 141s and 148s, so the new *cwnd* has been reduced by half, which is similar to NewReno. But, six error events are determined by NewZag as random packet losses at 7s, 23s, 124s, 136s 169s and 176s, so the new *cwnd* has been reduced to 4/5 of the current size. Compared to NewReno's blindly reducing *cwnd* by 1/2, NewZag decreases the window by a factor of 1/5. Thus, NewReno should take more time than NewZag to increase *cwnd* to a high value. On the contrary, if the error event comes from wireless random error, NewZag still remains *cwnd* on a high level and eventually increases the average TCP throughput.

The TCP modified strategies of NewZag are derived from [85], but have two amendments. Firstly, [85] uses Reno instead of NewReno as the TCP foundation algorithm. Compared with Reno, NewReno proposes modified fast retransmit and a fast recovery algorithm to handle the packet loss, so NewReno can have better TCP performance than Reno in most TCP connections [39]. Thus, NewZag inherits from NewReno but develops a more suitable algorithm.

Secondly, the TCP adjustments in NewZag are not the same as Internet Engineering Task Force (IETF) specifications but follow the implementations in the real Linux network stack. In [85], proposed TCP adjustments are based on IETF specifications, such as RFC 2581 [36] and RFC 2988 [51]. However, some mechanisms related to IETF specifications might be simplified for easier implementation in the real network environment. For instance, Linux TCP (kernel 2.4 and 2.6) did implement RFC 2581 (Reno), RFC 2582 [36] (NewReno), RFC 2988 and RFC 3168 [72] as the TCP congestion control mechanisms, but details differ in specification. According to [114], Table 5.1 explains the differences between IETF specifications and Linux implementations on mechanisms related to TCP congestion control. Since the NewZag adjustments are based on the real Linux TCP instead of the RFC specifications, demonstration results are closer to the realistic network.

Table 5.1: TCP congestion control related IETF specifications implemented in Linux. + = implemented, * = implemented, but details differ from specification.

Specification	Status
RFC 1323 (Perf. Extensions)	+
RFC 2018 (SACK)	+
RFC 2581 (Congestion control)	*
RFC 2582 (NewReno)	*
RFC 2861 (Cwnd validation)	+
RFC 2883 (D-SACK)	+
RFC 2988 (RTO)	*
RFC 3168 (ECN)	*

5.2.3 Integration of the Loss Differentiation Algorithm and SACK

The NewZag LDA can further cooperate with TCP Selective Acknowledgment (SACK) [41, 52] to improve the evaluation of packet loss number for three reasons. Firstly, the definition of packet loss numbers (n) was not described clearly in [82]. Secondly, the currently used TCP mechanisms (NewReno) on the Linux Internet do not provide explicit loss information to the TCP sender [114]. Thirdly, the LDA must deal with some "fake" loss packets. For instance, packet reordering is often a problem for the TCP sender because it cannot distinguish whether the missing ACKs are caused by a packet loss or by a delayed packet that will arrive later. Thus, the NewZag LDA must speculate which packets are the "real" loss in the network more and accurately calculate the packet loss number. If the feedback information on packet loss is not detailed, the accuracy of the proposed LDA might decrease. In Linux, TCP must determine which is the cause of packet loss out of four possibilities:

- The sender receives a triple ACK.
- A timeout occurs. In this case, *cwnd* is set to 2 and *ssthresh* is set to half of *cwnd* when the packet is lost.
- TX Queue is full.
- SACK detects a hole.

Thus, NewZag cooperates with TCP SACK in these four situations to detect the packet loss number in every error event. In contrary to New Reno, SACK provides the TCP sender with more accurate feedback on packet loss information during transmission [41]. This is because SACK implements an additional "SACK option" in the "Options field" of the TCP header. It invokes the most recently received packets and the most recently reported information on SACK packets. Thus, NewZag could have extra information to report the packet loss number in cooperation with SACK.

In the Linux operating system, integration of NewZag and SACK is simple and can easily detect the described four loss situations. This is because the SACK option has been implemented in many TCP protocol stacks, including Linux kernel version 2.2 and later. NewZag adjusts only small parts of NewReno and can fully cooperate with SACK without any extra effort.

The SACK function is turned on in default to operate with NewReno after the Linux kernel version 2.4. The demonstrations presented in the next section show the result of integrating SACK into NewReno and NewZag.

5.2.4 Millisecond Implementation into Linux TCP

The system time tick rate of Linux 2.4 kernel uses a 100Hz clock (10ms). It affects the evaluation accuracy since RTT, RTT_{mean} and RTT_{dev} are used as important factors inside the NewZag LDA. For instance, if the "real" RTT is between 50ms to 54ms, Linux TCP will present $50/10 = 5$ or $54/10 \cong 5$ as TCP measured RTT (round up or down). The same, if the "real" RTT is between 55ms to 59ms, Linux TCP will present $55/10 \cong 6$ or $59/10 \cong 6$ as TCP measured RTT variable. Because the clock graduation is too large, NewZag LDA might misjudge error types when NewZag classifies the TCP packet loss events in some specific situation. Thus, NewZag modifies Linux time functions and implements 1000Hz clock (1ms) in the Linux 2.4 kernel for high accuracy emulation results.

5.2.5 Summary of NewZag

The previous sections showed why NewZag might be considered as a better solution in the real world. NewZag determines the probable cause of packet loss and correspondingly chooses a suitable reaction. It derives its ideas from LDA, NewReno, and SACK. In order to better realize the status of real networks, it narrows down the Linux clock for more accurate results. However, the benefits of NewReno, such as initial slow start, additive increase, fast retransmit, fast recovery, computation of the retransmission timeout and the backoff algorithm still remain intact.

5.3 Performance Evaluation of TCP NewZag

This section presents verification of LDA as well as performance evaluation of TCP NewZag. NewZag not only achieves significant throughput improvements over NewReon, but can perform well when other TCP proposals share the same link.

5.3.1 Experimental Network Setup

Figure 5.6 shows the experimental network topology. Server 1 (S1) operates with Linux kernel 2.4 in which NewZag, NewReno and LTCP are set-up. Server 2 (S2) operates with Linux kernel 2.6 into which TCP Westwood, Veno, Vegas and NewReno are integrated. The SACK function at S1 and S2 is enabled. The wireless access point (AP) is ASUS-WL500g-Deluxe. The mobile host 1 (MH1) is a laptop, modelled Dell-D510. The mobile host 2 (MH2) is another laptop, Sony-Vaio-TX16. MH1 and MH2 operate with Linux kernel 2.6 and TCP NewReno is used as the default TCP receiver. MH1 and MH2 are connected to the router through AP. The queue size for each network interface "txqueuelen" is set at

1000. The IEEE 802.11b standard [5, 6] is implemented for the wireless links. The router is implemented for the use of netem [108] in which delay for wide area networks can be emulated. The DropTail queuing policy is used in the intermediate nodes. The wired links RTT between servers and AP are set at 50ms. The wireless link RTT between AP and MH is around 2.85ms. (averaged 50 times traced by "ping"). "ftp" connection is established between the simulated host and the laptop. Thus, the TCP behaviour can be observed through the ftp data transmission.

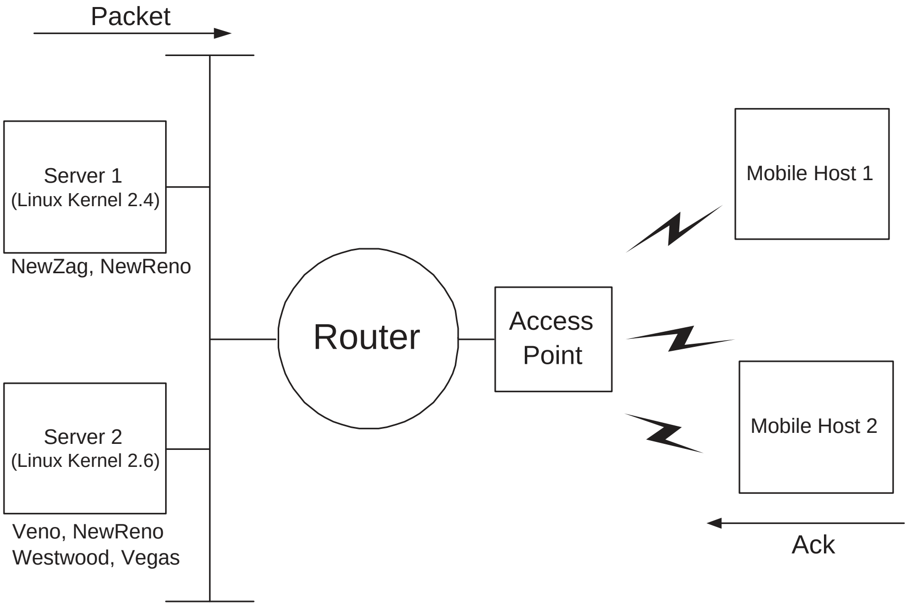


Figure 5.6: Experimental Network Topology.

The different kernel versions used in S1 and S2 are to compare NewZag experimental results with select TCP mechanisms. NewZag is constructed under Linux kernel 2.4 and the development for Linux kernel 2.6 is still in progress. Selected TCP proposals, such as Westwood, Veno and Vegas, are new modules supported only in Linux kernel 2.6. Thus, S2 is used to set-up the experimental environment of selected TCP versions.

5.3.2 Verification of Packet Loss Number Detection in NewZag

Verification of NewZag packet loss number is presented in this section. Figure 5.7 shows the *cwnd* relationship with the packet loss event. A single connection is set-up between server 1 and MH1, as shown in Figure 5.6. Double size of original TCP sender and receiver window are set up to create congestive loss during transmission. In order to receive some wireless-caused-packet loss, the wireless access point is located in a room at level 4 and MH1 is located in another room at level 3. Thus, the packet loss could come from the wired link or the wireless link in this topology.

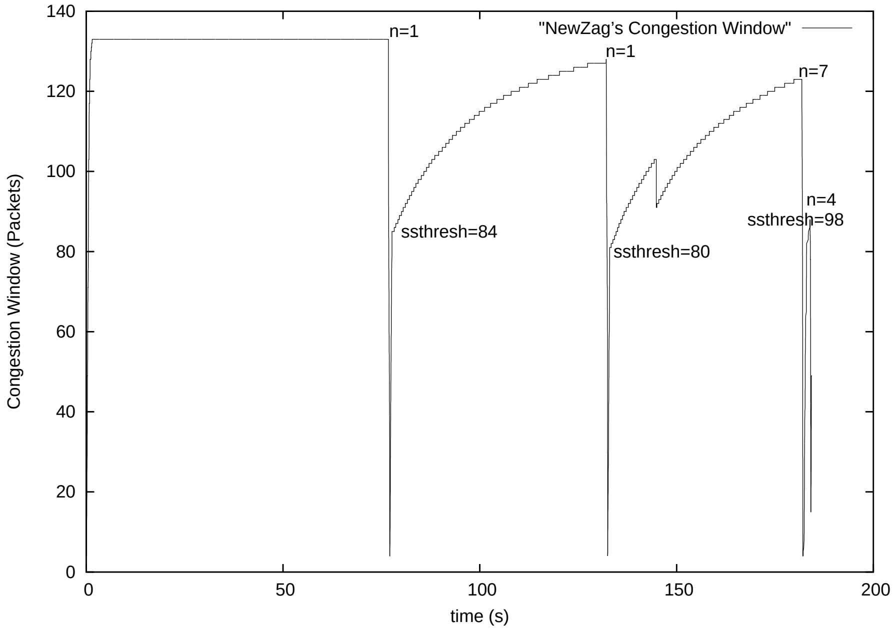
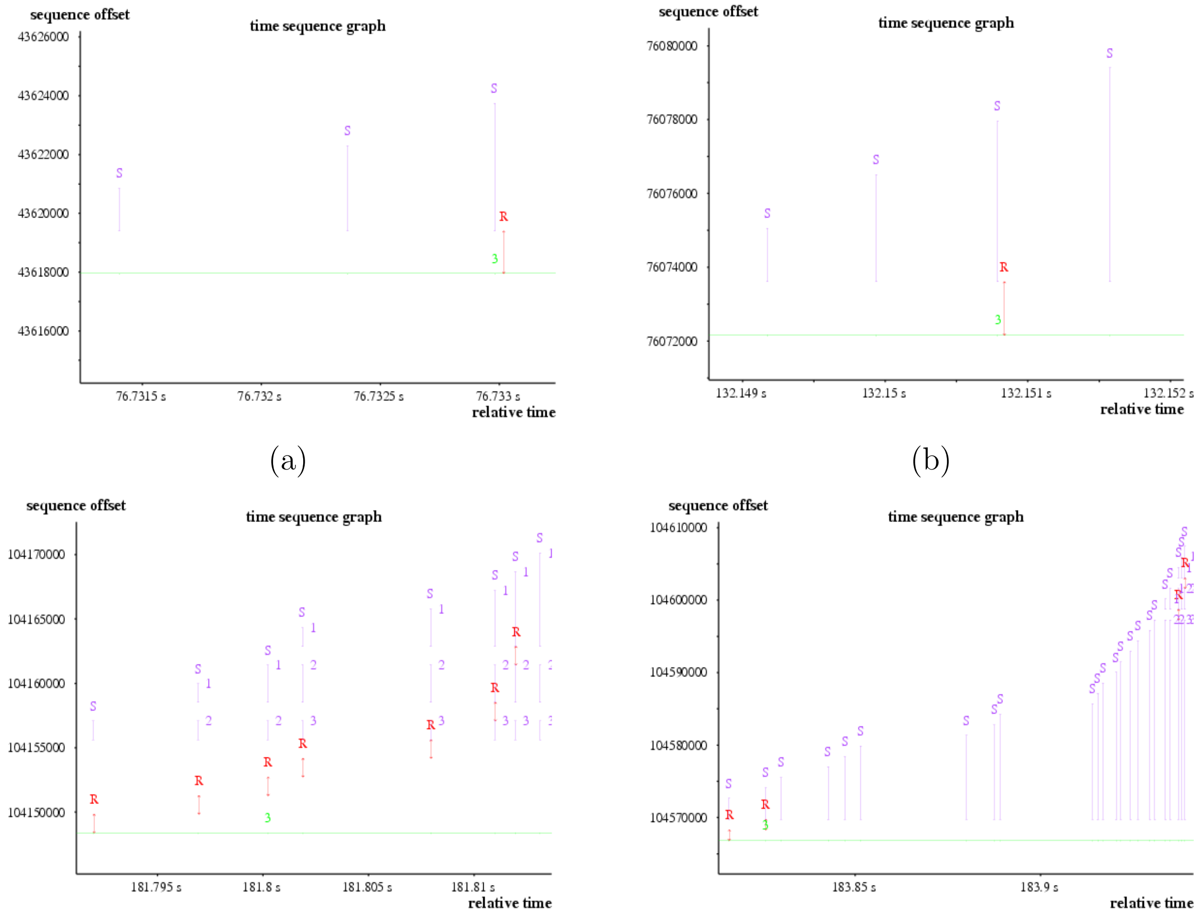


Figure 5.7: NewZag's integration with SACK. n is the number of packet loss. *ssthresh* means the slow-start threshold.

In this example, four packet loss events are detected by NewZag at around 76s, 132s, 181s and 183s. In cooperation with SACK, NewZag can correctly calculate that there is one packet loss (n=1) at 76s, one packet loss at 132s (n=1), seven packet losses (n=7) at 181s and four packet losses (n=4) at 183s. Thus, there is a total of 13 packet losses occurring in this transmission. At the same time, "tcpdump" is used to trace and record TCP behaviour on both the sender and the receiver sides. "tcptrace" is used to analyze the output file from the "tcpdump". Figure 5.8 shows the graphic output results from the "tcpdump". Label **R** signifies the retransmit packets. Figure 5.8 (a), (b), (c) and (d) display **R** in 1 time, 1 time, 7 times and 4 times, separately. It is clear that "tcptrace" shows the same packet loss numbers in each loss event as those calculated by NewZag. According to the emulation results, the loss counting mechanism of NewZag can achieve high accuracy if the loss event derives from the random packet loss (n=1 or n=2). But the loss event is caused by the queue dropping or heavy random packet loss(e.g. n is much higher than 3), NewZag might output unequal loss packet numbers as those counted from "tcpdump". However, this phenomenon will not affect the accuracy of NewZag, because it is unnecessary for NewZag to have precise packet loss numbers if n is greater than 3, as described in Section 5.2.1 above.

Figure 5.7 also shows the variation of *ssthresh.* When *cwnd* reaches 133, the first loss event (76s) is detected by the TCP sender. NewZag considers that the loss is caused by random packet loss, so the slow-start threshold is recalculated as $ssthresh = (current\ cwnd) * 4/5$. It is worth noting here that the new *ssthresh=84* comes from $4/5 * 105$ instead of $4/5 * 133$, because the Linux TCP should not choose the *cwnd* at loss occurrence but should select the last "seen" *cwnd* to calculate *ssthresh.* This also shows that Linux TCP differs from IETF



(a) (b) (c) (d)

Figure 5.8: Number of packet loss traced by "tcpdump". **R** presents the retransmit packets. Lines with an **S** on top means SACK blocks are found in ACK packets. **3** indicates that the received ACK packet was the triple duplicate ACKs, commonly used as the threshold to trigger the TCP fast retransmit/recovery algorithm.(a). One packet loss. (One **R** is presented).(b). One packet loss .(c). Seven packet losses. (d). Four packet losses.

specifications. The same occurred in NewReno in the Linux 2.4 kernel.

5.3.3 Packet Loss from the Wireless Links

In this simulation model, we try to control packet loss only occurring in the wireless part and follow the set packet loss rate. It is a real challenge to control a packet loss event in the real network environment, especially in specific links. Since the topology is constructed on the live network, it is still possible that small amounts of naturally lost packets occur during simulation, so simulation results are averaged over 10 simulations.

Two steps are used to control packet loss occurring only in the wireless links. The single-TCP connection is set up between MH1 (receiver) and server 1 (sender). Firstly, the queuing sizes of network nodes are adjusted to fit the current topology bandwidth and the sender and receiver window sizes keep setting as Linux kernel 2.4. Figure 5.9 shows the *cwnd* state when the packet loss rate is set to zero at the MH1. During transmission, *cwnd* is stable and finally holds at 69 until end of transmission, so no packet loss at either wired or wireless link is confirmed. Secondly, different packet loss rates (0%, 0.1%, 0.5%, 1% and 1.5%) are used at MH1 to simulate the operation of NewZag and NewReno. The uniform distribution bit error model is used and the TCP checksum is changed at the MH1 if the error generator determines that the packet should be dropped at the receiver.

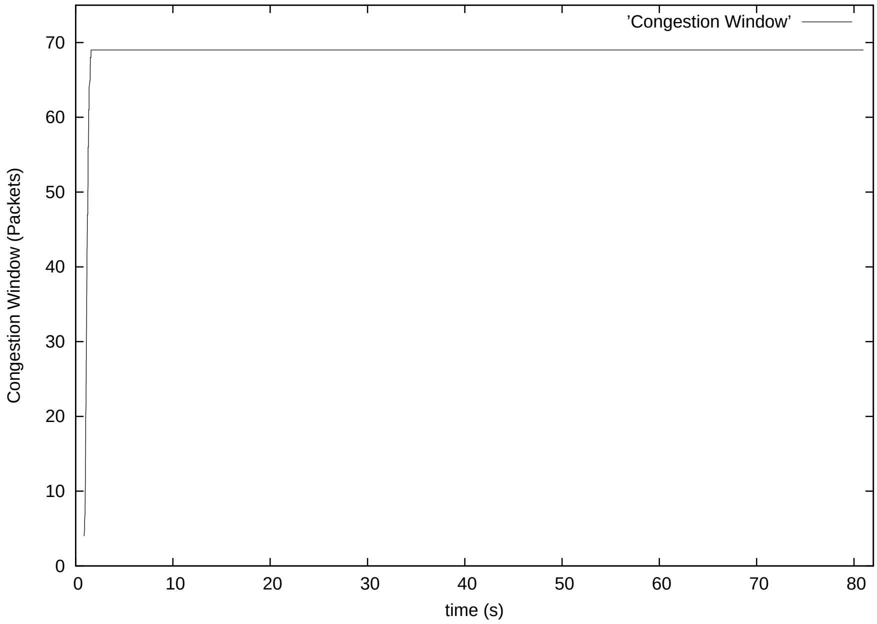


Figure 5.9: The congestion window with no packet losses.

Figure 5.10 shows throughput difference between NewZag and NewReno under different packet loss rates, results being averaged over 10 simulations. NewReno provides very similar results at Linux kernel 2.6 and kernel 2.4 in the same experiments, with different loss rates. The simulation of NewReno at kernel 2.6 is for contrast. This is because NewZag only adjusts NewReno mechanisms at Linux kernel 2.4. The similar result from NewReno simulation at kernel 2.4 and 2.6 means that NewZag can also perform similar output if NewZag is in future implemented into kernel 2.6. So, we will further compare NewZag with some TCP end-to-end proposals, which are only supported in Linux kernel 2.6.

Figure 5.10 shows that NewZag can perform better than Reno when different loss rates are set at MH1. But, the performance improvement of average TCP throughput can only reach 4% to 7%, because the NewZag LDA cannot accurately determine the loss type when lost packets are detected. Table 5.2 provides some important statistics. Two important observations can be made. The first is about the success rate of NewZag LDA. When the random loss rate is set at 0.1% at MH1, the TCP sender detects 33 loss events. In each loss event, NewZag detects only one lost packet (*n=1*). Thus, if $RTT_i < (RTT_{mean} - RTT_{dev})$, the loss event should be classified as random packet loss, as defined in NewZag LDA. But,

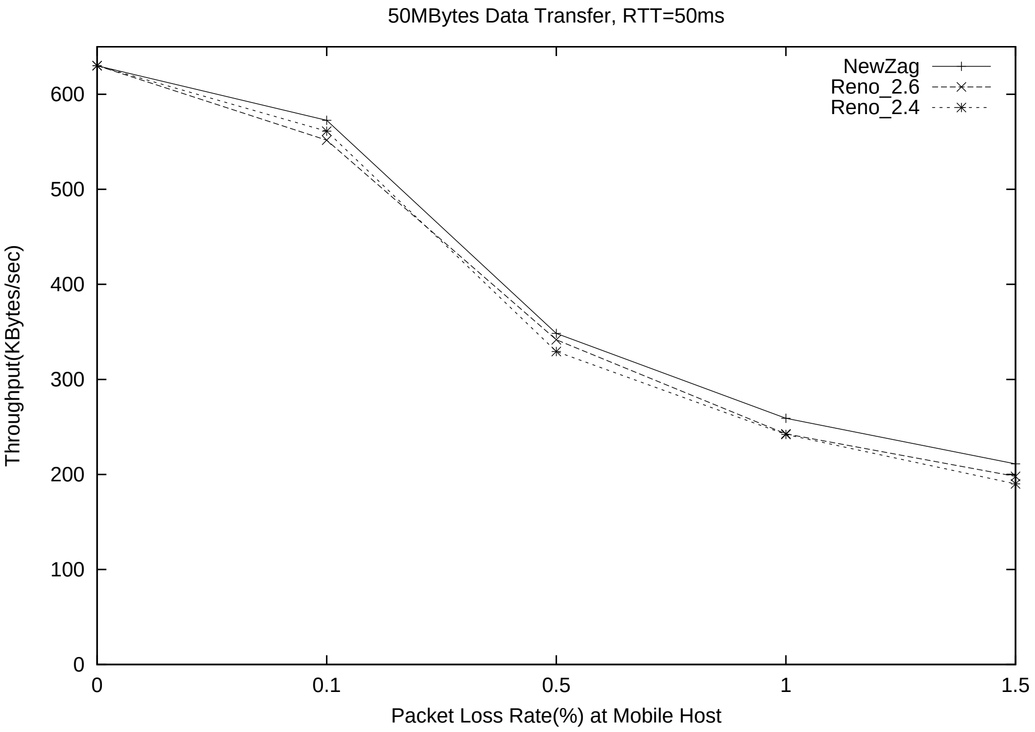


Figure 5.10: Simulation of NewZag and Reno in different packet loss rates on the MH. Reno_2.4 is the NewReno output results from Linux kernel 2.4. Reno_2.6 is the NewReno output result from Linux kernel 2.6.

NewZag LDA classifies 8 error events as random packet loss. It means that NewZag can only achieve around 24% success rate, because the random loss rate is set at only MH in this experiment. The second observation is that there are 18 error events happening in total, while $RTT < RTT_{mean}$, around 50%. However, only 8 error events are classified as random packet losses, fitting the condition $RTT_i < (RTT_{mean} - RTT_{dev})$. The rest of the 10 (18-8) error events are in the group of $RTT_i > (RTT_{mean} - RTT_{dev})$, defined as congestive errors by NewZag. The same phenomena occur at error rates 0.5%, 1% and 1.5%. The use of RTT and (RTT_{mean}) can provide current network conditions. If RTT is larger than (RTT_{mean}), there is a high probability that the packet loss comes from the congestive network (by queuing drop). On the other hand, if RTT is smaller than (RTT_{mean}), there is a high probability that the packet loss is caused by random packet loss. But NewZag LDA only considers the relationship of RTT and RTT_{mean} if and only if the packet loss number is equal to 3 (*n=3*) (see earlier section). However, we find that *n=3* only rarely occurs if there is only random wireless error, as shown in Table 5.2.

After the two observations above, NewZag LDA was slightly adjusted. The loss type is classified as random packet loss if:

$$[(\text{n} = 1,2,3) \quad \&\& \quad (RTT_i < RTT_{mean})]$$

Table 5.2: The observation of NewZag LDA in different loss rates at MH. The results are averaged from 10 simulations.

Loss rate at MH	0.1%	0.5%	1%	1.5%
Total error events (times)	33	180	330	487
numbers of packets lost(times):*n=1*	33	175	321	439
numbers of packets lost(times):*n=2*	0	4	20	46
numbers of packets lost(times):*n=3*	0	1	1	2
NewZag determines error events coming from Wireless Loss(times)	8	31	69	88
NewZag success rate	23%	17%	20%	18%
$RTT < RTT_{mean}$	18	80	140	200
$(RTT < RTT_{mean})$/Total error events	55%	44%	42%	41%

$$\text{OR } [(\text{n} > 3) \quad \&\& \quad (RTT_i < RTT_{mean} + RTT_{dev}/2)]$$

Figure 5.11 shows the difference between NewZag and NewReno under different packet loss rates after adjusting the methods. Other parameters are the same as the above experiment settings. It is shown that throughput of NewZag is consistently higher than that of NewReno for a range of random loss probabilities. In particular, at a loss rate of 1%, the throughput of NewZag (282.67KB/s) is 17% higher than that of NewReno(234.08KB/s). The remarkable results are mainly attributed to NewZag's refined multiplicative decrease algorithm that performs intelligent window adjustment based on the loss differentiation algorithm. Under low loss environments (random loss rate close to 0%), NewZag performs with around 2% higher throughput than NewReno because it behaves very much like NewReno during transmission, and random packet loss is not a significant factor any more. Thus, the modified NewZag LDA is much better at determining between congestion loss and random packet loss. We will further discuss modified NewZag with some TCP end-to-end proposals. Even though NewZag might not be able to recognize the random packet loss from all error events, it can still have the same performance as NewReno, since NewZag maintains most of NewReno's algorithms.

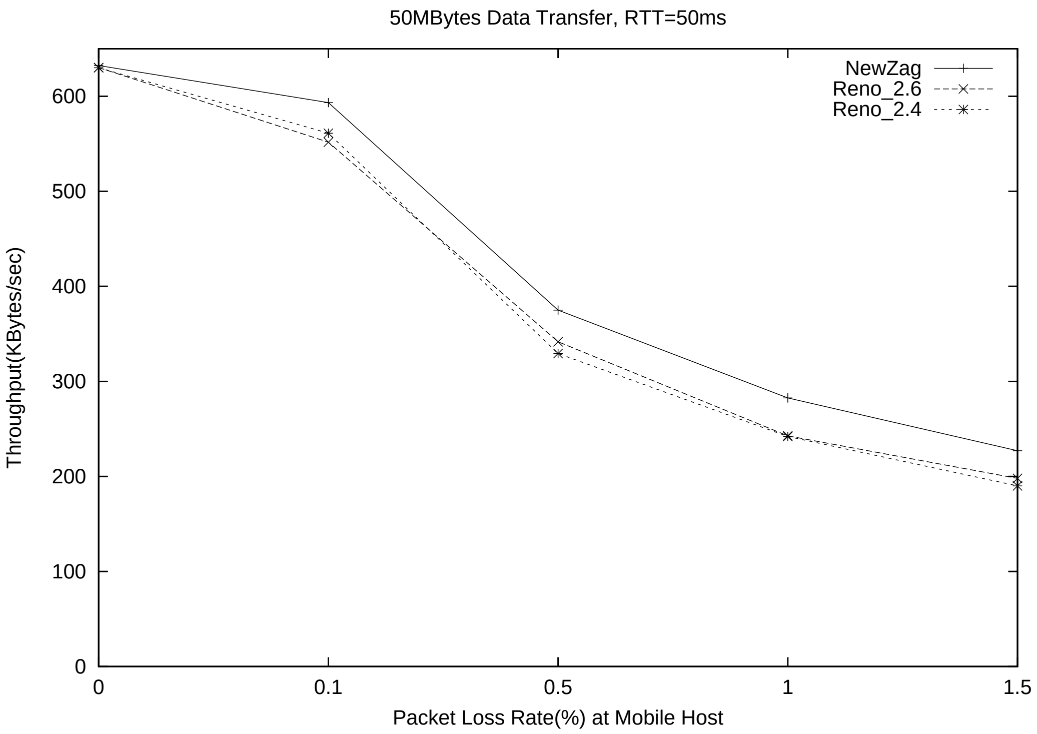


Figure 5.11: Simulation of adjusted NewZag LDA and Reno in different packet loss rates on the MH. Reno_2.4 is the output result from Linux kernel 2.4. Reno_2.6 is the output result from Linux kernel 2.6.

5.4 Experiments of Modified NewZag in Single and Competitive Connections

5.4.1 Single Connection Experiments

This section presents single-TCP connection results (i.e., only one of the source-destination pairs is configured, as shown in Figure 5.6). The TCP sender and receiver parameters are kept the same as the original settings in Linux kernel 2.4. The packet loss rate is only set at MH1. Other parameters are the same as in the experiment discussed in Section 5.3. Figure 5.12 shows NewZag experiments with some different ene-to-end TCP versions.

Figure 5.12 shows the differences between NewZag, Westwood, Veno and Vegas under different packet loss rates. The throughput of NewZag is shown to be consistently higher than that of Westwood, Veno and Vegas for the range of random loss probabilities. In particular, at a loss rate of 0.1%, NewZag(593.37KB/s) is 7% higher than that of Westwood(552.05KB/s), 19% higher than that of of Veno(480.44KB/s) and 65.1% higher than that of Vegas(206.9KB/s).

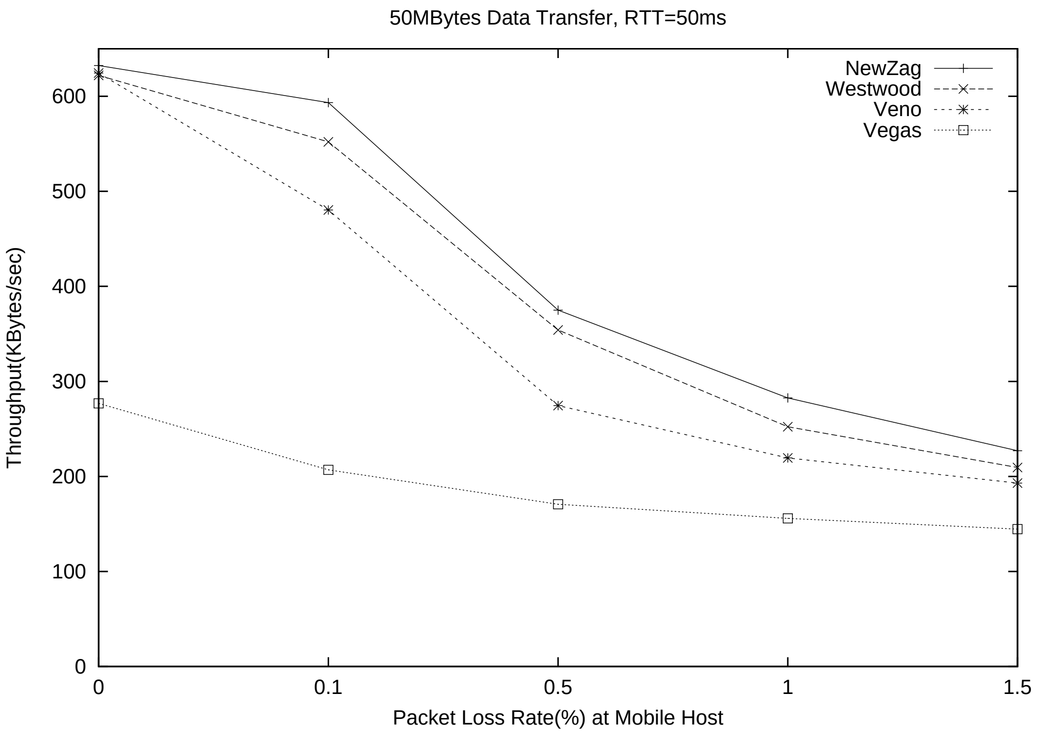
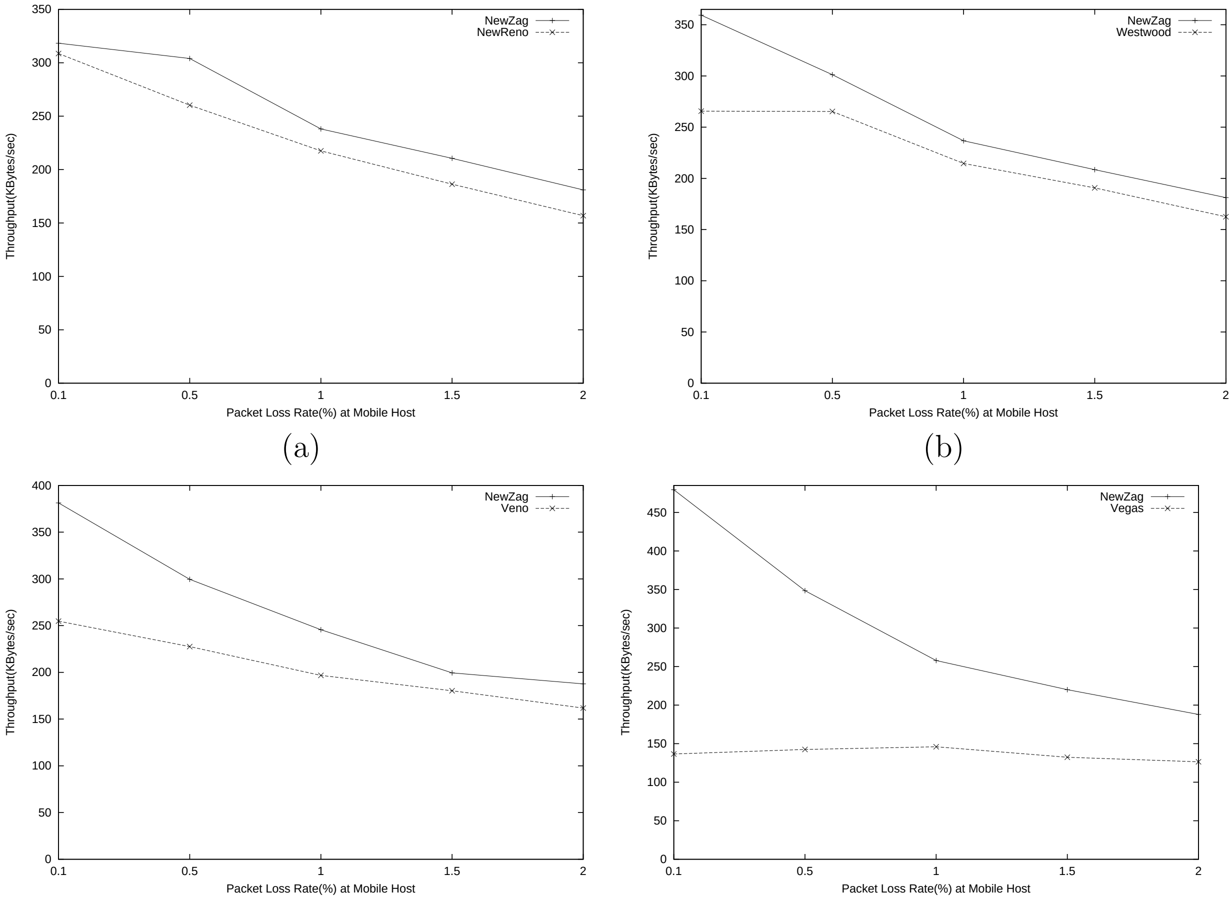


Figure 5.12: Comparison of NewZag, Westwood, Veno and Vegas in different packet loss rates on the MH.

5.4.2 Experiments Involving Two Co-existing Connection

The experiments in this section, sought to understand the influences of NewZag as it performed on competing flows with other TCP mechanisms. The experimental topology involves two co-existing TCP connections. In Figure 5.6, two connections set up a mix of NewZag with NewReno, Westwood, Veno and Vegas separately to share a common link. To be precise, we take NewZag as sharing common links with NewReno as an example, as shown in Figure 5.6 (a). Different packet loss rates: 0.1%, 0.5%, 1%, 1.5%, 2% are examined, separately. Results at each packet loss rate are averaged over 10 experiments. For the first 5 experiments, the NewZag connection is set-up between MH1 and S1, and the NewReno connection is set up between MH2 and S2. For the last 5 experiments, the NewZag connection is set-up between MH2 and S1, and NewReno connection is set up between MH1 and S2. In each experiment, two connections start data transmission at the same time and maintain transmission for 100s. The average throughput is measured as *Total received data/ 100s.* Other parameters are the same as the experiment settings discussed in Section 5.3.

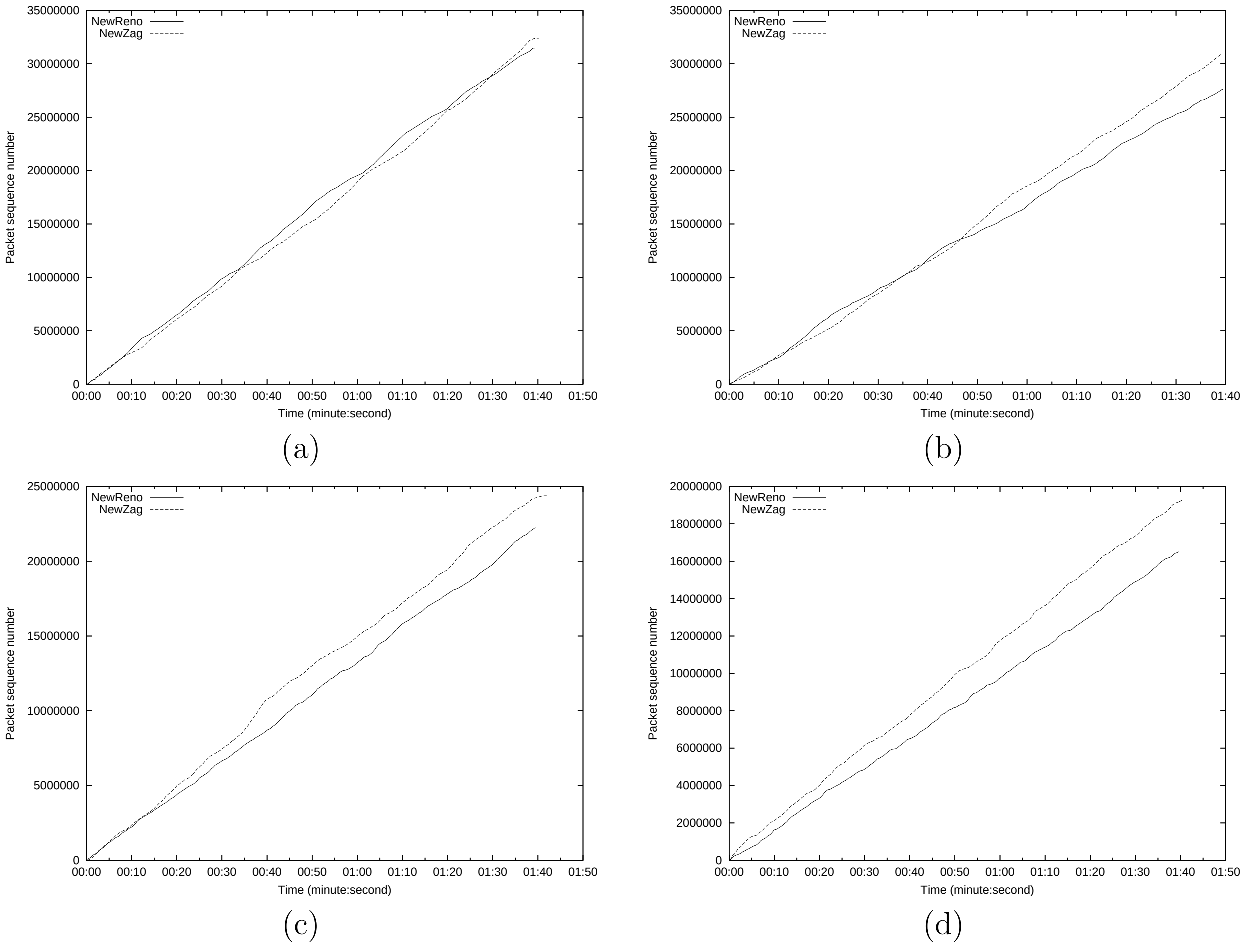
In Figure5.13, three key points can be found. The first observation is about the influence of NewZag when it shares common links with other TCP links. In experiments with NewReno, NewZag throughput is consistently higher than that of NewReno for a range of random loss probabilities. The throughput of NewZag is about 15% higher, as shown in Figure5.13 (a). When NewZag shares a common link with Westwood or Veno, it can perform with about



(a) (b)

(c) (d)

Figure 5.13: Two connections are set-up to share a common link in 100s transmission. (a). NewZag shares the common link with NewReno. (b). NewZag shares the common link with Westwood. (c). NewZag shares the common link with Veno. (d). NewZag shares the common link with Vegas.



(a)

(b)

(c)

(d)

Figure 5.14: NewZag shares a common link with NewReno in 100s transmission. (a). Random packet loss rate 0.1%. (b). Random packet loss rate 0.5% (c). Random packet loss rate 1% (d). Random packet loss rate 2%

8% higher throughput than that of Westwood, and about 15% higher than Veno. Thus, the performance of NewZag is still improved even though it shares common links with other competitive links.

The second observation is about the low throughput of Vegas when it shares a common link with NewZag. When Vegas connection shares a link with NewZag, Vegas throughput is lower than NewZag, which causes unfairness connection. Because the NewZag connection uses most of the buffer space and the TCP Vegas connection backs off from interpreting this as a sign of network congestion, Vegas automatically decreases the *cwnd*. In this situation, the packet loss rate is not the important factor causing the low throughput of Vegas. However, the unfairness transmission does not come from the NewZag mechanism. It derives from the original design of Vegas. When Vegas shares a common link with other TCP versions, the low throughput of Vegas is also shown. This phenomenon is also discussed in [120], in which Reno shares a link with Vegas. Since Veno inherits Vegas' backbone queuing condition. Veno cannot perform as expected at every loss rate, as addressed in [85]

The third observation is about fairness. Figure 5.14 shows the results of NewZag and

NewReno sharing a common link at different random packet loss rates. Figure 5.14 (a) shows the sequence number growing between two connections with a random loss rate of 0.1%. Because of the low loss rate, NewZag competes with NewReno in an almost fair manner, since the sequence numbers in the two connections are roughly the same. Figure 5.14 (b) shows the experimental result with 0.5% random loss rate. NewZag competes similarly to NewReno from 0s to 50s and performs better from 50s to 100s. Figure 5.14 (c) and (d) shows the experimental results with random loss rate of 1% and 2%, separately. The NewZag sequence number is consistently higher than that of NewReno for 100s transmission due to high error rates. In summary, NewZag does not adversely affect coexisting NewReno at low loss rates, so we can conclude that NewZag is compatible with NewReno. Besides, NewZag performs better than NewReno when a high random loss rate occurs.

5.4.3 Comparisons Between NewZag and ZigZag

This sub-section compares NewZag and ZigZag. NewZag references end-to-end loss differentiation algorithm presented in ZigZag. However, the emulation comparisons between NewZag and ZigZag are not discussed in this chapter. The main reason is that the TCP modification codes of ZigZag are implemented in ns2, which were not supported inside the Linux kernel. However, the experimental results discussed in this chapter are all based on Linux system, so the Linux experiment on ZigZag is not included. It is still possible to test the ZigZag experiment on live Linux emulation in the future, but it will require implementation of ZigZag into the Linux kernel or as Linux network modules. Afterwards, the reliability of ZigZag Linux code must been verify and that will take time. Thus, some high level comparisons of ZigZag and NewZag are discussed below:

- Effects on current Internet: ZigZag uses an $ROTT$ variant to evaluate LDA, so the modification of current TCP must be made on both the sender and the receiver. Compare with ZigZag, NewZag modifications are implemented only on the TCP sender side, so the effect of current Internet status is minimum.
- Modified LDA : According to live experiment discussed in Section 5.3.3, ZigZag LDA might not efficiently differentiate loss types. Thus, NewZag proposes a new modified NewZag LDA, which is much better at determining between congestion loss and random packet loss.
- TCP adjustments after LDA: NewZag did not clearly discuss the TCP behaviour after its LDA distinguishes the loss types. But, TCP adjustments have been implemented in NewZag after LDA. Afterwards, the TCP behaviour has also been discussed in live emulation and compared with some selected wireless TCPs.

5.5 Summary

Random packet loss in wireless networks could lead to performance degradation in end-to-end TCP connections. A novel TCP version, called TCP NewZag, is proposed to distinguish random packet loss from congestion loss. Numerous demonstrations have been made in experimental networks. The experimental results show that TCP NewZag can achieve significant improvement compared with other versions of wireless TCP.

Four standpoints are used to conclude TCP NewZag:

1. Compatibility: Compatibility refers to whether TCP NewZag is compatible with popular current TCP on the Internet (NewReno). It means that NewZag does not cause any disadvantageous effects to NewReno. When NewZag competes with NewReno in a common link, NewZag does not "steal" bandwidth from NewReno. NewZag only contributes to the enhancement of throughput when high random packet loss occurs in wireless links.

2. Deployability: Deployability refers to whether TCP NewZag is easy to deploy over the existing Internet. An end-to-end TCP connection could cross many intermediate nodes. Thus, it would be perfect if the change requirement at the intermediate nodes is little or unnecessary, so the effect of TCP modifications can be limited at two end hosts. For the two end nodes (sender and receiver), it is preferred that change in one side is required. Furthermore, backwards compatibility of previous network systems is preferred, so that the the cost of corresponding Internet modification can be limited. The simple modification at only the sender side makes TCP NewZag easily deployable on the live Internet. It is backwards compatible with current network stacks, so the implementation of NewZag can reduce the effect of current network systems.

3. Efficiency: Efficiency refers to whether TCP NewZag can achieve performance improvement in TCP in wireless access networks. From experimental results, we not only compares NewZag with NewReno, but also compare NewZag with other wireless TCPs. Remarkable results show that NewZag would be a worthwhile object of future TCP research.

4. Flexibility: Flexibility refers to whether TCP NewZag can still perform well if it is implemented on the Internet in various environments. It is challenging to declare that NewZag is more flexible than other versions of wireless TCP. Further tasks can be completed in future work in this research area.

Although the idea of TCP NewZag references several previous proposals with some adjustments, it was not a straightforward task to implement NewZag in the Linux network environment due to the gap between theory and the live network. NewZag re-designs the clock unit inside the kernel to increase the estimate of three RTT relative variables. From numerous experimental results, it is found that the cooperation of NewZag and SACK can precisely feedback packet loss numbers to the TCP sender. Even though NewZag focuses on the Linux kernel 2.4, we believe that it can still perform well in Linux kernel 2.6, from experimental results.

Chapter 6

Conclusions and Future Work

This chapter mainly concludes the thesis work, and enumerates several research problems that can be addressed in future work.

6.1 Summary of Contributions

The research aims to study how standard and proposed TCP mechanisms fit in wireless environments and then tries to enhance TCP performance in the heterogeneous network. Generally speaking, the contributions of this research are:

- Studying TCP challenges:

  TCP was initially designed for use in a wired network. TCP has weaknesses on a heterogeneous network. In order to enhance TCP performance, its TCP behaviour on the real network needs to be determined. Hence, TCP challenges and possible solutions on the heterogeneous network were discussed. Wireless TCP proposals can be classified into three groups: link layer proposals, split-connection and end-to-end proposals. They have also been studied. Subsequently, it was found that most proposals have their strengths, but some also have weakness. For instance, the link layer proposals could cause extensive delay variation at the IP level. The split-connection proposals could violate the end-to-end semantic, and optimization issues in intermediate nodes must be considered. Thus, a better solution might be end-to-end proposals that maintain basic TCP semantics, which could focus modifications on the sender side. Therefore, an end-to-end proposal that is backwards compatible could be a better answer, to limit the adaptation of the current network framework.

- Producing a new TCP simulator:

  A novel prototype of a TCP simulator, LTCP, was developed. A simulator can be used to understand TCP behaviour, so further analysis and discussion can be made. LTCP provides several features that cannot be easily achieved by traditional network simulators. LTCP enables simulation of complex network topologies on only one Linux PC. LTCP uses real Linux TCP/IP stacks to imitate simulation results. The statistics from simulations are nearly realistic, because the whole simulation environment is close to real network. The Linux network interface and utilities are naturally included as tools of the simulator. Since LTCP is based on the Linux TCP stack, it does not matter what kind of network application is being used. No alteration against the simulator is required. This feature is useful for further TCP based research.

- Producing a new emulator:
  An emulator was produced that was based on LTCP. This emulator is a prototype of a wireless TCP emulator. It can be used to monitor TCP behaviour in a live wireless link by building a demonstrated environment (call a framework). In this research, in order to get accurate results, a simple and economic framework was built. Large numbers of computing processors are no longer necessary. With the emulator and the built framework, the complicated emulation process is simplified, while the emulation might be closer to reality. Since this emulator is built on top of a Linux TCP/IP stack, it is easier for users to monitor the interactions and details during the emulation. This elaboration might be useful for those who want to study more than the simulation results. The emulator not only produces a realistic reflection, but also enables functions to simulate novel protocols, which can be used in end-to-end data delivery on a mixture of wireless and wired networks. The wired network topology is simulated by LTCP. Thus, the user can easily separate factors to configure wireless networks.

- Proposing a new mechanism NewZag:
  A NewZag mechanism was proposed to enhance TCP performance in the heterogeneous network. NewZag proposes a new loss differentiation algorithm (LDA) to distinguish the congestion loss from random packet loss. NewZag also modifies the sender side protocol of NewReno without changing the receiver side protocol. But, the benefits inherit from NewReno still remain intact. NewZag maintains TCP end-to-end semantic without breaking the model of Open Systems Interconnection (OSI), so the effect of current Internet is minimized. The NewZag features of compatibility, deployability and efficiency have been discussed separately. TCP NewZag was shown to perform well in different considerations.

- Experimental evidence to prove NewZag to have better performance in the heterogeneous network:
  NewZag improves network performance in comparison with several well-known wireless TCPs. In IEEE 802.11b wireless networks with 1% random packet loss rate, NewZag is 17% higher than NewReno, 7% higher than Westwood, 19% higher than Veno and 65.1% higher than Vegas in single connection experiments. When NewZag shares a common link with select wireless TCP versions, NewZag can still perform remarkable

performance in the heterogeneous network. The throughput of NewZag is about 15% higher than NewReno, about 8% higher than Westwood and about 15% higher than Veno. In experimental work, NewZag demonstrated satisfactory throughput without interfering other concurrent TCP connections, including NewReno, Veno and Westwood.

6.2 Future Work

This section describes possible extensions to research.

6.2.1 LTCP Future Work

LTCP will improve in future versions in some areas, as introduced below.

- Distributed architecture: In this architecture, each component of LTCP can be run on separate machines, in what is called the “multi-machine” mode. A “job dispatcher agent” will be designed as an interface between the LTCP and simulation topology. The agent is used to manage a simulation job and dispatch it for LTCP, which carries an advantage. When a large network topology is simulated, LTCP may take a long time to simulate complex simulation parameters. To overcome this in a distributed architecture, a simulation job can be run on two or more LTCP PCs to speed up the simulation.

- Support for simulation in various networks: Simulation on modern wireless protocols will be supported by LTCP, for instance IEEE 802.11 (g) WiMAX networks and Wireless Sensor Network (WSN), which have been implemented in different areas in recent years. Moreover, the support of multi-interface mobile nodes equipped with multiple heterogeneous wireless interfaces is also an important objective. This type of mobile nodes will become common and widely used type in real life. For instance, users can choose the most cost effective network to connect to the Internet at any time and at any location.

- Support for handover simulation: MHs traveling in different BSs during transmission is common. The support of handover simulation can give researchers an inside look at their simulation results.

- Support for simulation of Internet Protocol Version 6 (IPv6): Simulation on IPv6 will be supported by LTCP. IPv6 is the “next generation” protocol designed to replace the current version Internet Protocol, IP Version 4 (IPv4). IPv6 mechanisms have also been implemented into Linux kernel. If LTCP and emulator discussed in Chapter 4 can support IPv6 simulation, it can help users study TCP behaviour in the future.

6.2.2 NewZag Future Work

NewZag implements and modifies some mechanisms from ZigZag. The comparison of two algorithm might be discussed in the future. NewZag is implemented and simulated on the wired linked servers. Further discussion might be interesting. For instance, if the NewZag server is linked as a mobile host and clients are at wired link, would it make any difference? If two or more mobile hosts (all NewZag enabled) connect via wireless channels (as ad-hoc network), what would happen? Since mixture connection, wired connection or wireless connection might have different types of errors, NewZag might function differently in these simulations. NewZag implementation in different wireless networks, such as WiMAX and WSN. They should be worthy to study.

Bibliography

[1] K. Romer and F. Mattern, “The Design Space of Wireless Sensor Networks,” *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, Dec. 2004.

[2] T. Haenselmann, *An FDL’ed Textbook on Sensor Networks.* [Online]. Available: http://www.informatik.uni-mannheim.de/ haensel/sn_book/

[3] “Bluetooth Web.” [Online]. Available: http://www.thewirelessdirectory.com/

[4] D. M. Gilster, *Bluetooth End to End.* Wiley, Mar. 2002.

[5] D. L. Lough, T. K. Blankenship, and K. J. Krizman, “A Short Tutorial on Wireless LANs and IEEE 802.11,” vol. 5, no. 2, 1997.

[6] “The Institute of Electrical and Electronics Engineers website.” [Online]. Available: http://standards.ieee.org/getieee802/802.11.html

[7] “GSM Association.” [Online]. Available: http://www.gsmworld.com/about/index.shtml

[8] C. Everhart, L. Mamakos, and R. Ullmann, “New DNS RR Definitions,” RFC 1183, Oct. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1183.txt

[9] 3GPP, “GSM Enhanced Full Rate Speech Processing Functions: General Description,” The 3rd Generation Partnership Project, Technical Specification TS06.51, Dec. 1997. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/0651.htm

[10] 3GPP, “GPRS Tunnelling Protocol GTP across the Gn and Gp Interface,” The 3rd Generation Partnership Project, Technical Specification 29.060v5.8.0, Dec. 2003. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/29060.htm

[11] A. Malis, D. Robinson, and R. Ullmann, “Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode,” RFC 1356, Aug. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1356.txt

[12] Information Sciences Institute, University of Southern California, “Ineternet Protocol,” RFC 791, Sept. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[13] Y. Rekhter, T. Li, and Editors, "An Architecture for IPv6 Unicast Address Allocation," RFC 1887, Dec. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1887.txt

[14] "International Telecommunication Union: Worldwide Mobile Cellular Subscribers to Reach 4 Billion Mark Late 2008." [Online]. Available: http://www.itu.int/newsroom/press_releases/2008/29.html

[15] M. Frodigh, P. Johansson, and P. Larsson, "Wireless Ad Hoc Networking : The Art of Networking Without a Network," *Ericsson Review*, no. 4, pp. 248–263, 2000.

[16] I. 802.16-2004, "IEEE Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed Broadband Wireless Access Systems," Tech. Rep., Oct. 2004.

[17] I. 802.16-2005, "IEEE Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed Broadband Wireless Access Systems for Mobile Users," Tech. Rep., Dec. 2005.

[18] M. Tubaishat and S. Madria, "Sensor Networks: an Overview," *IEEE Potentials*, vol. 22, no. 2, pp. 20–23, Apr. 2003.

[19] Y. S. I.F. Akyildiz, W. Su and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communication Magazine*, Aug. 2002.

[20] "RFID.org." [Online]. Available: http://www.aimglobal.org/technologies/rfid/

[21] M. Carson and D. Santay, "NIST Net: a Linux-based Network Emulation Tool," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003.

[22] M. Allman, A. Caldwell, and S. Ostermann, "ONE: The Ohio Network Emulator," Ohio University," Technical Report, 1996.

[23] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *ACM Computer Commun. Review*, vol. 27, no. 1, pp. 31–41, 1997.

[24] P. Danzig, Z. Liu, and L. Yan, "An Evaluation of TCP Vegas by Live Emulation," in *Proceedings of ACM SIGMetrics '95*, 1995.

[25] L. Brakmo and L. Peterson, "Experiences with Network Simulator," in *Proc. of SIGMETRICS*, Philadelphia, USA, May 1996, pp. 80–90.

[26] K. Lien and J. Reeve, "A High Accurate and Component Based Network Emulator for Simulation of Complex Heterogeneous Network Topology," in *Postgraduate Research Conference in Electronics, Photonics, Communications and Networks, and Computing Science (PREP 2005)*, Lancaster,UK, Mar. 2005, pp. 153–154.

[27] K. Lien and J. Reeve, "A TCP/IP Network Emulator," in *International Symposium on Telecommunications, IST2005*, Shiraz, IRAN, Sept. 2005, pp. 659–664.

[28] K. Lien and J. Reeve, “A TCP/IP Network Simulator,” in *International Conference on Information Technology and Integration of Manufacturing and Business(ITIMB)*, Changhua,Taiwan, Aug. 2006, pp. 319–328.

[29] K. Lien and J. Reeve, “A Comparison of Methods to Improve TCP Performance over Wireless Networks,” in *International Conference on Information Technology and Integration of Manufacturing and Business(ITIMB)*, Stockholm, Sweden, Aug. 2006, pp. 448–497.

[30] K. Lien, J. Reeve, and Y. Lee, “Improving TCP Performance Over Wireless Networks,” in *International Symposium on Telecommunications, IST2008*, Telecom, IRAN, Aug. 2008, pp. 424–428.

[31] “Internet Pioneers Cerf and Kahn To Receive ACM Turing Award.” [Online]. Available: http://campus.acm.org/public/pressroom/press_releases/2_2005/turing_2_14_2005.cfm.

[32] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet.* USA: Addison-Wesley, July 2002.

[33] J. Postel, “Transmission Control Protocol,” RFC 793, Sept. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[34] R. Braden, “Requirements for Internet Hosts - Communication Layers to IP,” RFC 1122, Oct. 1989. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[35] V. Jacobson, “Congestion Avoidance and Control,” in *Proc. of ACM SIGCOMM’88*, Stanford, CA, Aug. 1988, pp. 314–329.

[36] M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control,” RFC 2581, Apr. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[37] M. Allman, “TCP Congestion Control with Appropriate Byte Counting (ABC),” RFC 3465, Feb. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3465.txt

[38] S. Floyd, “HighSpeed TCP for Large Congestion Windows,” RFC 3649, Aug. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[39] S. Floyd and T. Henderson, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” RFC 2582, Apr. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2582.txt

[40] S. Floyd, T. Henderson, and A. Gurtov, “The NewReno Modification to TCP’s Fast Recovery Algorithm,” RFC 3782, Aug. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3782.txt

[41] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[42] S. Floyd, J. Mahdavi, M.Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," RFC 2883, July 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2883.txt

[43] E. Blanton, M. Allman, K. Fall, and L. Wang, "A conservative selective acknowledgment (sack)-based loss recovery algorithm for tcp," RFC 3517, Jan. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3517.txt

[44] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[45] W. R. Stevens, *TCP/IP Illustrated, The Protocols (APC).* Reading: Addison-Wesley, Jan. 1994, vol. 1.

[46] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," Tech. Rep., Apr. 1990.

[47] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, Jan. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2001.txt

[48] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," RFC 2861, July 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2861.txt

[49] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows," RFC 3742, Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3742.txt

[50] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 3390, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3390.txt

[51] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," RFC 2988, Nov. 2000. [Online]. Available: http://www.faqs.org/rfcs/rfc2988.html

[52] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.

[53] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144, Feb. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1144.txt

[54] E. H. Inamura, E. G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks," RFC 3481, Feb. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3481.txt

[55] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," RFC 3042, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3042.txt

[56] R. Atkinson and S. Floyd, "IAB Concerns and Recommendations Regarding Internet Research and Evolution," RFC 3869, Aug. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3869.txt

[57] R. Jain, "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Network," *ACM Compututer Communication*, vol. 19, no. 5, pp. 56–71, Oct. 1989.

[58] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug. 1993.

[59] M. Taferner and E. Bonek, *Wireless Internet Access Over GSM and UMTS.* Springer-Verlag Berlin and Heidelberg GmbH & Co. K, Oct. 2001.

[60] G. Giambene and D. Miorandi, "Performance Evaluation of Scalable TCP and Highspeed TCP over Geostationary Satellite Links," in *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, vol. 25, Changhua,Taiwan, May 2005, pp. 2658– 2662.

[61] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, Feb. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1323.txt

[62] 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.

[63] R. Caceres and L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 5, pp. 850–857, June 1995.

[64] H. Elaarag, "Improving TCP Performance over Mobile Networks," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 357–374, Sept. 2002.

[65] S. Nanda, R. Ejzak, and B.T.Doshi, "A Retransmission Scheme for Circuit-Mode Data on Wireless Links," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 8, pp. 1338 – 1352, Oct. 1994.

[66] F. Fitzek, B. Rathke, M. Schlger, and A. Wolisz, “Simultaneous MAC-Packet Transmission in Integrated Broadband Mobile System for TCP,” in *ACTS SUMMIT 1998*, Rhodos, Greece, June 1998, pp. 580–586.

[67] G. Carle, F. Fitzek, and A. Wolisz, “Combining Transport Layer and Link Layer Mechanism for Transparent QoS Support of IP based Applications,” in *IP Quality of Service for Wireless and Mobile Networks (IQWiM99)*, Aachen, Germany, Apr. 1999.

[68] A. Bakre and B. Badrinath, “I-TCP: Indirect TCP for Mobile Hosts,” in *Proc. of the 15th Intl. Conference on Distributed Computing Systems*, Vancouver, Canada, May 1995, pp. 136–143.

[69] A. Bakre and B. R. Badrinath, “Handoff and System Support for Indirect TCP/IP,” in *Second Usenix Symposium on Mobile and Location-Idependent Computing*, Michigin, USA, Apr. 1995.

[70] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, “Improving TCP/IP Performance over Wireless Networks,” in *Proc. of the First Annual International Conference on Mobile Computing and Networking*, California, USA, Dec. 1995, pp. 2–11.

[71] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, “Improving Performance of TCP over Wireless Networks,” in *Proc. of the 17th International Conference on Distributed Computing Systems (ICDCS ’97)*, Washington, USA, May 1997, pp. 365–373.

[72] K. Ramakrishnan, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168, Sept. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[73] R. Yavakar and N. Bhagawat, “Improving end-to-end Performance of TCP over Mobile Internetworks,” in *Proc. IEEE MCSA’94*, CA, USA, Dec. 1994, pp. 146–152.

[74] D. D. Clark, “The Design Philosophy of the DARPA Internet Protocols,” in *SIGCOMM*, vol. 18, no. 4. CA, USA: ACM, Aug. 1998, pp. 106–114. [Online]. Available: citeseer.ist.psu.edu/clark88design.html

[75] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, “TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links,” in *Proceedings of ACM Mobicom 2001*, Rome, Italy, 2001, pp. 16–21.

[76] C. Casetti, M. Gerla, S. Mascolo, M. Sansadidi, and R. Wang, “TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks,” *Wireless Networks*, vol. 8, pp. 467–479, 2002.

[77] G. Yang, R. Wang, M. Y. Sanadidi, and M. Gerla, “Performance of TCPW BR in Next Generation Wireless and Satellite Networks,” in *ICC 2003*, Alaska, May 2003.

[78] E. Lengliz, H. Touati, F. Kamoun, and M. Y. Sanadidi, "Experimentations towards TCP Westwood Application ," in *Adhoc Mobile Networks Med-Hoc Net 2003*, Tunis, Tunisia, June 2003.

[79] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "Using Adaptive Rate Estimation to Provide Enhanced and Robust Transport over Heterogeneous Networks," in *Proc. 10th IEEE International Conference on Network Protocols 2002*, Paris, France, Nov. 2002, pp. 206–215.

[80] K. J. Astrom and B. Wittenmark, *Computer-Controlled Systems: Theory and Design.* N. J: Prentice Hall: 3 Edition, 1996.

[81] S. Biaz and N. H. Vaidya, "Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times at the Receiver," in *IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, TX, USA, Mar. 1999, pp. 10–17.

[82] S. Cen, P. Cosman, and G. Voelker, "End-to-end Differentiation of Congestion and Wireless Losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 703–717, Oct. 2003.

[83] Y. T. Tobe, Y. Molano, and H. A. Ghosh, S. Tokuda, "Achieving Moderate Fairness for UDP Flows by Path-status Classification," in *Local Computer Networks, 2000. LCN 2000. Proceedings. 25th Annual IEEE Conference on*, Tampa, FL, USA, Nov. 2000, pp. 252–261.

[84] A. Boukerche, G. Jia, and R. W. N. Pazzi, "Performance Evaluation of Packet Loss Differentiation Algorithms for Wireless Networks," in *Proc. of the 2nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, Chania, Greece, Oct. 2007, pp. 50–52.

[85] C. P. Fu and S. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, Feb. 2003.

[86] N. Samaraweera, "Non-Congestion Packet Loss Detection for TCP Error Recovery using Wireless Links," *IEEE Proceedings of Communications*, vol. 146, no. 4, pp. 222–230, Aug. 1999.

[87] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *Computer Communications Review, ACM SIGCOMM*, vol. 19, no. 45, pp. 56–71, Oct. 1989.

[88] V. Tsaoussidis and H. Badr, "TCP-probing: Towards an Error Control Schema with Energy and Throughput Performance Gains," in *Proceedings of the 2000 International Conference on Network Protocols*, Osaka, Japan, Nov. 2000, pp. 12–21.

[89] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for Wireless IP Communications," *IEEE JSAC*, vol. 22, no. 4, pp. 747–756, May 2004.

[90] F. Peng, S. Cheng, and J. Ma, "An Effective Way to Improve TCP Performance in Wireless/mobile Networks," in *EUROCOMM 2000. Information Systems for Enhanced Public Safety and Security. IEEE/AFCEA*, Munich, Germany, May 2000, pp. 250–255.

[91] G. Montenegro *et al.*, "Long Thin Networks," RFC 2757, Jan. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2757.txt

[92] M. Allman and A. Falk, "On the Effective Evaluation of TCP," *ACM Computer Communication Review*, vol. 29, no. 5, pp. 59–70, Oct. 1999.

[93] V. Paxson and S. Floyd, "Why We Don't Know How to Simulate the Internet," in *Proc. of the 29th Conference of Winter Simulation*, Atlanta, USA, Dec. 1997, pp. 1037–1044.

[94] S. McCanne and S. Floyd, "ns-LBNL Network Simulator." [Online]. Available: http://www.isi.edu/nsnam/ns/.

[95] A. Varga, "OMNeT++ Object-oriented Discrete Event Simulation System." [Online]. Available: http://www.omnetpp.org/

[96] S. Wang and H. Kung, "A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulator," in *Proc. IEEE INFOCOM'99*, New York, USA, Mar. 1999, pp. 1134–1143.

[97] S. Wang *et al.*, "The Design and Implementation of the NCTUns 1.0 Network Simulator," *Computer Networks*, vol. 42, no. 2, pp. 175–197, June 2003.

[98] M. Zec and M. Mikuc, "Real-Time IP Network Simulation at Gigabit Data Rates," in *Proc. of the 7th Intel. Conference on Telecommunications*, Zagreb, Croatia, June 2003, pp. 235–242.

[99] X. W. Huang, R. Sharma, and S. Keshaw, "The ENTRAPID Protocol Development Environment," in *Proc. IEEE INFOCOM'99*, vol. 4, New York, USA, Mar. 1999, pp. 1107–1115.

[100] S. Keshav, "REAL: A Network Simulator," Department of Computer Science," Technical report, 1998.

[101] "The ns-3 Network Simulator." [Online]. Available: http://www.nsnam.org/.

[102] "Opnet inc." [Online]. Available: http://www.opnet.com

[103] M. Kojo *et al.*, "Seawind: A Wireless Network Emulator," in *Proc. of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, Aachen, Germany, Sept. 2001.

[104] D. Mahrenholz and S. Ivanov, "Real-Time Network Emulation with ns-2," in *Proc. of 8-th IEEE International Symposium on Distributed Simulation and Real Time Applications*, Budapest Hungary, Oct. 2004, pp. 29–36.

[105] M. Zec, "Operating System Support for Integrated Network Emulation in IMUNES," in *Proc. of 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure*, Boston, USA, Oct. 2004.

[106] "The Universal TUN/TAP Driver Website." [Online]. Available: http://vtun.sourceforge.net/tun/

[107] "The iproute2 Website." [Online]. Available: http://linux-net.osdl.org/index.php/Iproute2

[108] "The Netem Website." [Online]. Available: http://linux-net.osdl.org/index.php/Netem

[109] [Online]. Available: http://jarok.cs.ohiou.edu/software/tcptrace/download.html

[110] H. Lee, S. Lee, and Y. Choi, "The Influence of the Large Bandwidth-Delay Product on TCP Reno, NewReno, and SACK," in *Proc. of the 15th International Conference on Information Networking*, Oita, Japan, Jan. 2001, pp. 327–334.

[111] C. Chung, C. P. Fu, and S. Liew, "Improvements Achieved by SACK Employing TCP Veno Equilibrium-oriented Mechanism over Lossy Networks," in *Proc. of EURO-CON'2001, Int. Conf. Trends in Communications*, Bratislava, Slovakia, July 2001, pp. 202–209.

[112] A. Varga, "OMNeT++ Object-oriented Discrete Event Simulation System User Manual." [Online]. Available: http://www.omnetpp.org/external/doc/html/usman.php.

[113] W. R. Stevens and G. A. Wright, *TCP/IP Illustrated: The Implementation (APC)*. Reading: Addison-Wesley, Mar. 1995, vol. 2.

[114] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP," in *Proc. of the FREENIX Track: 2002 USENIX Annual Technical Conference*, California, USA, June 2002, pp. 49–62.

[115] V. Paxson, "On Calibrating Measurements of Packet Transit Times," in *Proc. of SIG-METRICS*, Wisconsin, USA, 1998, pp. 11–21.

[116] S. Moon, P. Skelly, and D. Towsley, "Estimation and Removal of Clock Skew from Network Delay Measurements," in *Proc. IEEE INFOCOM'99*, vol. 1, New York, USA, Mar. 1999, pp. 227–234.

[117] K. Anagnostakis, M. Greenwald, and R. Ryger, "cing: Measuring Network Internal Delays Using Only Existing Infrastructure," in *Proc. IEEE INFOCOM'03*, vol. 3, San Francisco, USA, Mar. 2003, pp. 2112–2121.

[118] D. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 245–254, 1995.

[119] D. Mills, "Network Time Protocol," RFC 1305, Mar. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1305.txt

[120] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," in *Proc. IEEE INFOCOM'99*, vol. 3, New York, USA, Mar. 1999, pp. 1556–1563.

[121] "xplot." [Online]. Available: http://www.xplot.org/.

[122] "jPlot." [Online]. Available: http://www.tcptrace.org/jPlot.

Appendix A

Installation Information of LTCP

This appendix gives an installation guide to LTCP. The source code of "LTCP" can be required from email "kwlien@gmail.com". The first section introduces the directories included in LTCP. The second section shows the LTCP installation steps.

A.1 Introduction of LTCP Directories

Seven directories are included in LTCP, as follows:

- Applications directory: Many useful network applications and utilities are included in this directory. The applications can be used in simulation. For instance, "ftpd" and "ftp" are used to generate traffic flows. "ping" is used to test round trip time. The utilities can be used to analyze simulation data. For instance, "tcptrac" is used to generate different types of output information and graphs for further analysis. "tcptrace" is further discussed in Appendix B.
- EventScheduler directory: The event scheduler of the network simulator is stored in this directory. A detailed description of the LTCP event scheduler was given in Chapter 3.
- KernelModifications directory: All kernel modified codes are located in this directory. The user must copy these codes into the kernel to replace the original kernel source codes. The steps for doing this are detailed in the next section.
- CompileKernel directory: This directory is the place for users to start learning about Linux kernel compiling. Many documents regarding Linux kernel compiling are located in this directory.
- SimulationExample directory: This directory is the place for users to start LTCP tutorials. A few LTCP simulation examples are described to teach users how to make simulations based on LTCP.

- Document directory: Two different documents are included in this directory. The first comprises some LTCP related papers. The second introduces the steps towards LTCP installation, as detailed in the next section.
- Version directory: The LTCP version number is shown in this directory.

A.2 Installation Guide of LTCP

A.2.1 Introduction of LTCP Installation Steps

There are two steps towards installing LTCP into Linux OS. Firstly, users must copy the LTCP source code into the Linux kernel source code. In this step, the user must make sure that the kernel source code has been previously installed in the system. The user can check the directory "/usr/src" in advance. In this directory, there should be a symbol link named linux-2.X with links to the real directory linux-2.X.X-X. The directory linux-2.X.X-X is the location of the kernel source code. If this directory exists, the following commands can be used to put the LTCP source code into the Linux kernel. Otherwise, the user must install the Linux kernel source code in advance before starting. Secondly, the user must re-compile the Linux kernel source code to build a new kernel. The process is described in the first part of this section and in the second part of the next section.

```
# set NEWKERNELPATH=/usr/src/'uname -r'
# cp drivers/net/* $ { NEWKERNELPATH }/drivers/net
# cp include/linux/* $ { NEWKERNELPATH }/include/linux
# cp include/net/* $ { NEWKERNELPATH }/include/net
# cp ipv4/* $ { NEWKERNELPATH }/net/ipv4
```

A.2.2 Building a New Linux Kernel

As described in the last section, when the LCTP source codes are put into the Linux kernel, a new kernel must be compiled to complete the installation. The steps for new kernel compilation described in this section might only be used in RedHat 9.0.

1. The user must change the directory to the location of the Linux kernel source. If there is no special description, the following operational directory can be found in this location.

   ```
   # cd /usr/src/linux-2.4
   ```

2. Before compiling the new kernel, the user must clean unnecessary files from this directory.

   ```
   # make mrproper
   ```

3. The user should copy the original Linux config file as the new kernel config file.

   ```
   # cp configs/kernel-2.4.20-i686.config .config
   ```

4. The user must make some basic settings before kernel compiling. For instance, the user must include a TUN/TAP device for use with LTCP.

   ```
   # make xconfig
   # select "Load Configuration from File"
   # input ".config" as the Enter filename
   # select "OK"
   # select "Network device support"
   # select 'y' for "Universal TUN/TAP device driver support"
   # go back to ``Main Menu"
   # select "Save and Exit"
   ```

5. The user should then complete the following five steps to compile a new kernel.

   ```
   # make dep
   # make clean
   # make bzImage
   # make modules
   # make modules install
   ```

6. The user should copy the new kernel into the appropriate directory and give the new kernel an appropriate name. Then, we edit the “menu.lst” file and put new kernel as the new boot up selection. The “menu.lst” is the menu for multiple boot up selection. For safety, the user should save the original menu selection and insert a new selection for LTCP.

   ```
   # cp arch/i386/boot/bzImage /boot/LTCP
   # cp System.map /boot/System.map-LTCP
   # cd /boot
   # cd grub
   # gedit menu.lst
   ```

7. Reboot your PC and select new kernel to execute Linux. Then, the LTCP can be executed in this Linux system.

Appendix B

Tcptrace Tutorial

This appendix gives an short tutorial on “tcptrace” [109]. The tutorial only focus on the content relative to the thesis. A detailed description of “tcptrace” can be found in the **TCP-TRACE Manual**, which is downloadable from {http://www.tcptrace.org/manual.html}. The first section introduces basic usage of “tcptrace”. The second section displays and introduces some graphs generated from “tcptrace”.

B.1 Basic Usage

“tcptrace” is a TCP analysis tool. It was written by Dr.Shawn Ostermann. It is maintained later by his students and members of the Internetworking Research Group (IRG) at Ohio University. “tcptrace” can be run on a network “dumpfile”. The `dumpfile` is a text file containing network traffic captured from the links. “tcptrace” is aware of various network dumpfile formats: such as tcpdump, snoop, etherpeek, netm and ns. “tcptrace” can also be passed multiple command-line options to perform various tasks. Users can use `tcptrace -h` to get a brief descriptions of various command-line options. When “tcptrace” is run on a dumpfile, it generates output file as: [109]:

```
Bell:/Users/kwlien> tcptrace newzag.dmp


1 arg remaining, starting with 'newzag.dmp' Ostermann's tcptrace --

version 6.4.5 -- Fri July 15, 2006


90 packets seen, 90 TCP packets traced elapsed wallclock time:

0:00:00.037900, 3100 pkts/sec analyzed trace file elapsed time:

0:00:12.180796 TCP connection info:
```

```
1: imweb.im.ctu.edu.tw:61000- imbt.im.ctu.edu.tw:ssh (a2b) 45> 45<

(complete)


2: imweb.im.ctu.edu.tw:61001 - tw.yahoo.com:http (c2d) 12> 15<

(complete)
```

As shown above, “tcptrace” is run on dumpfile `newzag.dmp`. The initial lines explain that the file “tcptrace” is processing `newzag.dmp`, the version of “tcptrace”, and when compiled. The next line gives information that 90 packets were seen in the dumpfile and all the 90 TCP packets (in this case) were traced. The next line gives information about the `elapsed wallclock time` i.e., the time “tcptrace” took to process the dumpfile, and the average speed, in packets per second, taken for processing. The following line indicates the `trace file elapsed time` i.e., the duration of packet capture of the dumpfile calculated as the duration between the capture of the first and last packets.

The subsequent lines indicate the two TCP connections traced from the dumpfile. The first connection was seen between machines `imweb.im.ctu.edu.tw` at TCP port `61000`, and `imbt.im.ctu.edu.tw` at TCP port `ssh` (22). The second connection was seen between machines `imweb.im.ctu.edu.tw` at TCP port `61001`, and `tw.yahoo.com` at TCP port `http` (80). “tcptrace” uses a labeling scheme to refer to individual connections traced. In the above example, two connections are labeled `a2b` and `c2d`, respectively. For the first connection, 45 packets were seen in the a2b direction (imweb.im.ctu.edu.tw ==> imbt.im.ctu.edu.tw) and 45 packets were seen in the b2a direction (imbt.im.ctu.edu.tw ==> imweb.im.ctu.edu.tw). The two connections are reported as `complete`, indicating that the entire TCP connection was traced i.e., SYN and FIN segments opening and closing. TCP connections may also be reported as `reset` if the connection was closed with an RST segment, or `unidirectional` if traffic was seen flowing in only one direction.

The above output generated by “tcptrace” can also be generated with the `-b` option. In the above example, “tcptrace” looked up names (imbt.im.ctu.edu.tw, for example) and service names (http, for example), involving a Domain Name System (DNS) name lookup operation. Such name and service lookups can be turned off with the `-n` option to make the “tcptrace” process faster. The `-s` option can be used if name lookups are needed (`imbt` instead of `imbt.im.ctu.edu.tw` for example).

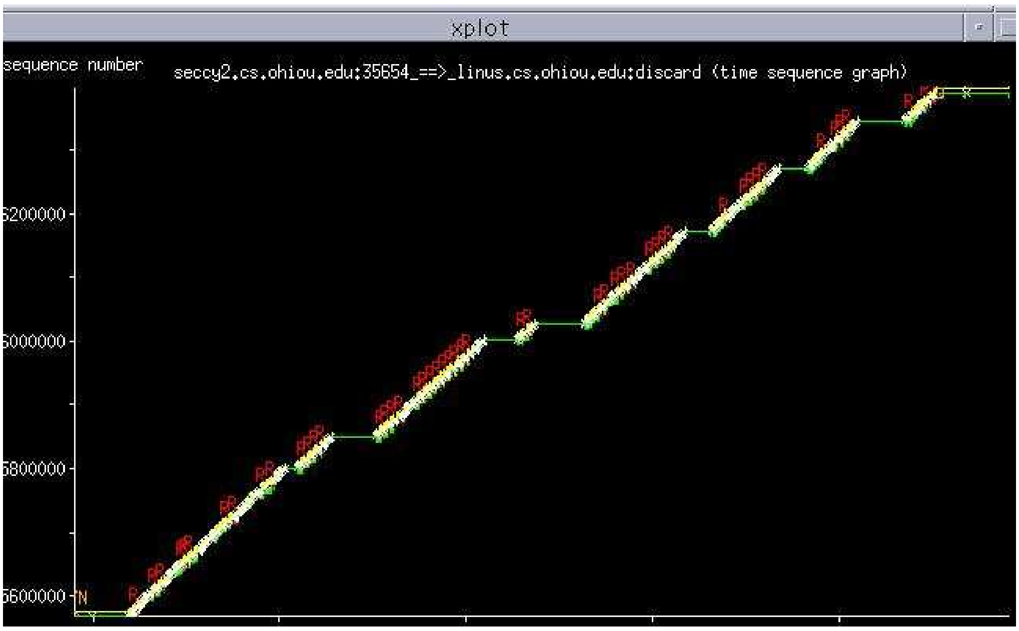

xplot
sequence number
seccy2.cs.ohiou.edu:35654_==>_linus.cs.ohiou.edu:discard (time sequence graph)
6200000
6000000
5800000
5600000
N
R

Figure B.1: Time Sequence Graph #1. (Source: TCPTRACE Manual.)

B.2 Graphing

"tcptrace" can generate six different types of graphs, illustrating various parameters of a TCP connection. These graphs can be viewed with Tim Shepard's xplot program [121] or with the Java version of the same program called jPlot [122], developed by Avinash Lakhiani.

The graphs and the options for "tcptrace" that generate them, are explained below. "tcptrace" leaves *.xpl data files in the working directory when the graphing options are given, and the data files can be viewed with the xplot program as in

```
xplot a2b_tsg.xpl
```

B.2.1 Time Sequence Graph

Time Sequence graphs show the general activity and events that occur during the lifetime of a connection. It can be generated with the `-S` option. These graphs are named as X2Y_tsg.xpl. A sample Time Sequence graph is shown in Figure B.1.

The X-axis represents time and the Y-axis represents sequence number space. The slope of the curve gives the throughput over time. Figure B.2 is a section from this graph (zoomed in with xplot), illustrated in the following features [109].

- **Green Line** keeps track of the ACK values received from the other endpoint.
- **Yellow Line** tracks the receive window advertised from the other endpoint. (It is drawn at the sequence number value corresponding to the sum of the acknowledgment

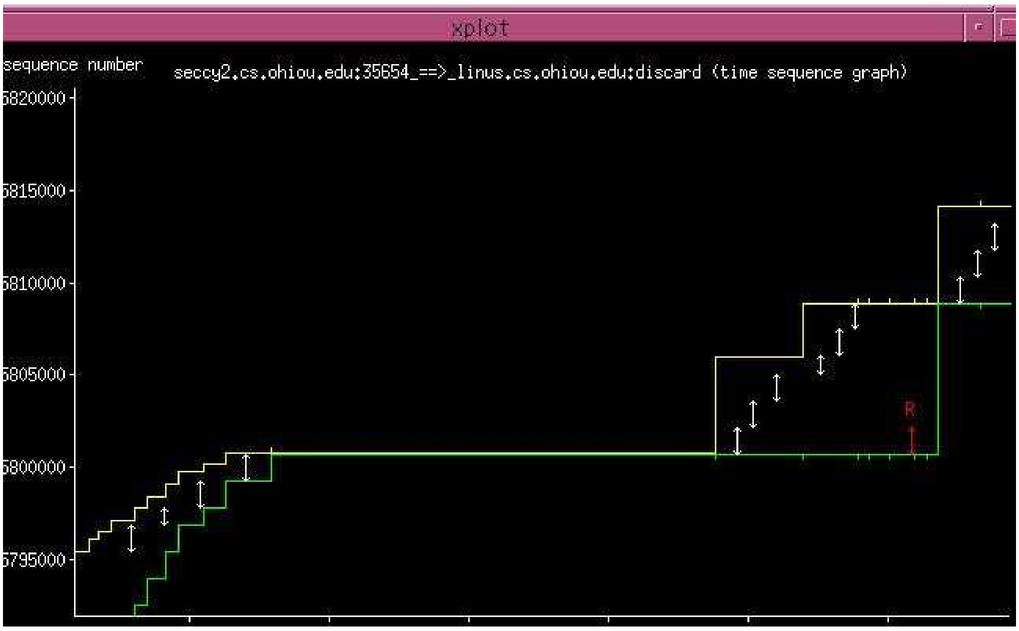

xplot
sequence number
seccy2.cs.ohiou.edu:35654_==>_linus.cs.ohiou.edu:discard (time sequence graph)
5820000
5815000
5810000
5805000
5800000
5795000
R

Figure B.2: Time Sequence Graph #2. (Source: TCPTRACE Manual.)

number and the receive window advertised from the last ACK packet received.)

- **Little Green Ticks** track the duplicate ACKs received.
- **Little Yellow Ticks** track the window advertisements that were the same as the last advertisement.
- **White Arrows** represent segments sent. The up and down arrows represent the sequence numbers of the last and first bytes of the segment respectively.
- **Red Arrows (R)** represent retransmitted segments with the up and down arrows similarly representing the sequence numbers of the last and first bytes of the segment.

Further zooming into the beginning of the connection with xplot is shown in Figure B.3. Here, the **SYN** marks the sequence number and the time when a SYN packet was sent.

The graph shown in Figure B.4 is a section of a TCP connection being closed. Here,

- **FIN** marks a FIN segment sent in the direction.
- **RST_IN, RST_OUT**: When a RST segment is sent, a **RST_OUT** is marked in the graph, and a **RST_IN** is marked in the Time Sequence graph of the opposite direction of the connection.
- **Little crosses (x)**: These are segments sent with zero TCP data payload (the down and up arrows of the segment coincide, giving rise to a cross).

SACK [41, 42] blocks found in ACK packets are represented as purple lines with an **S** on top as shown in Figure B.5.

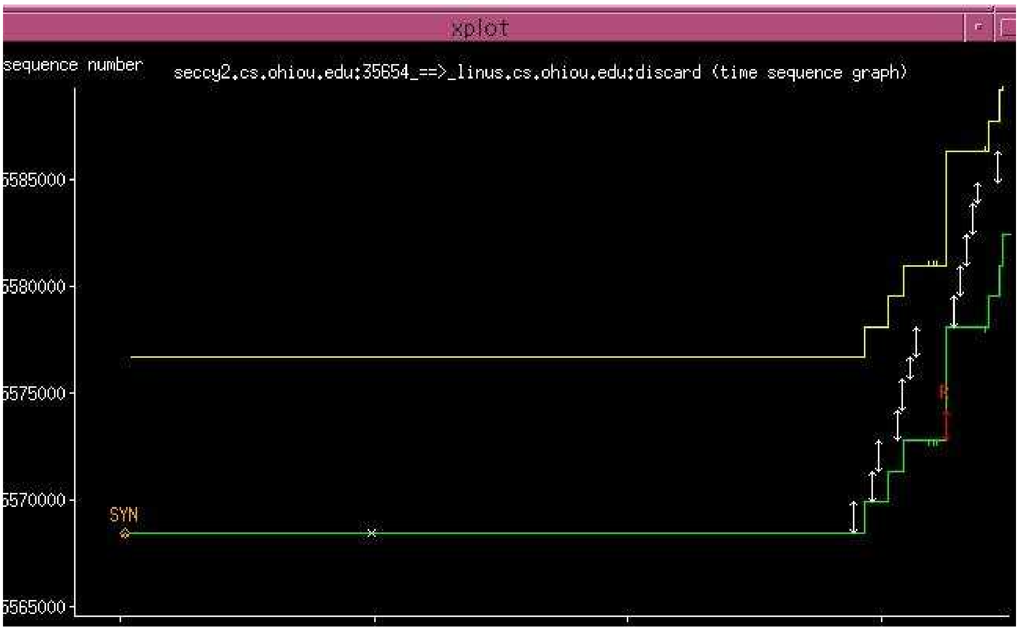

xplot
sequence number
seccy2.cs.ohiou.edu:35654_==>_linus.cs.ohiou.edu:discard (time sequence graph)
5585000
5580000
5575000
5570000
5565000
SYN
R

Figure B.3: Time Sequence Graph #3. (Source: TCPTRACE Manual.)

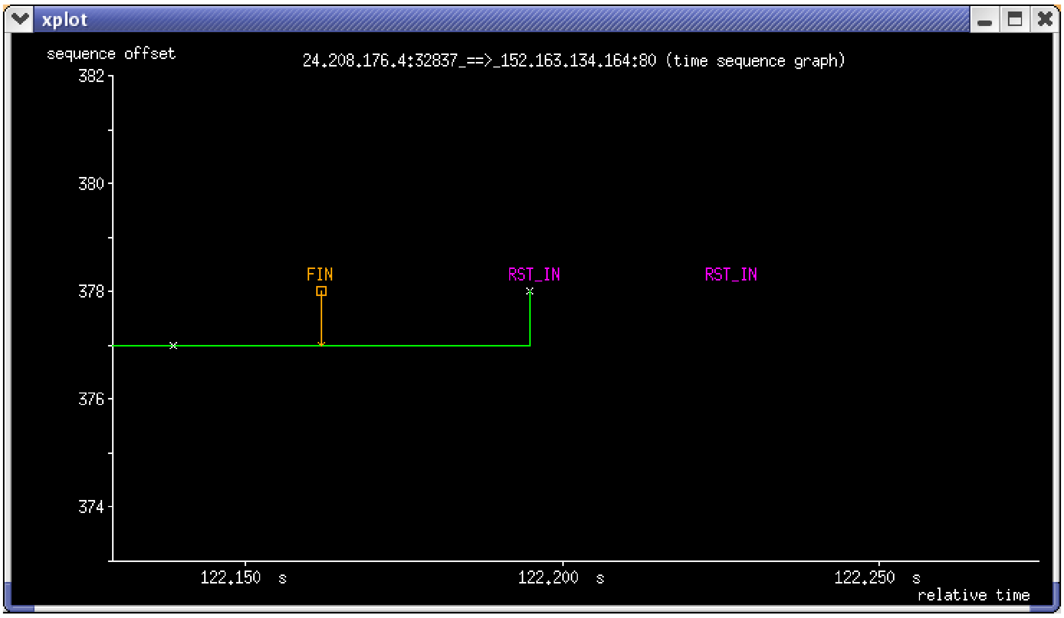


Figure B.4: Time Sequence Graph #4. (Source: TCPTRACE Manual.)

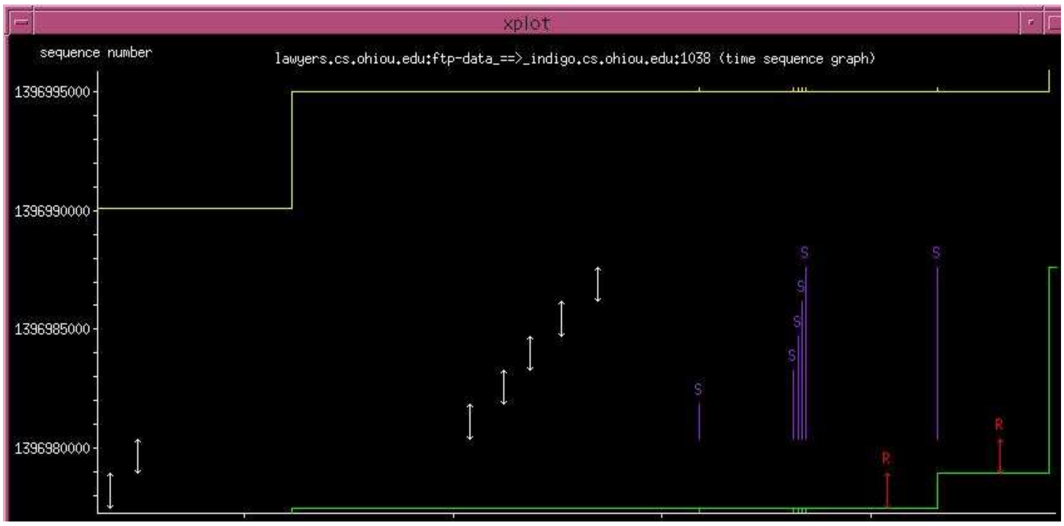


Figure B.5: SACK blocks. (Source: TCPTRACE Manual.)

PUSH segments, i.e., TCP segments sent with the PUSH flag set are represented with a Diamond in place of the up arrow as shown in Figure B.6.

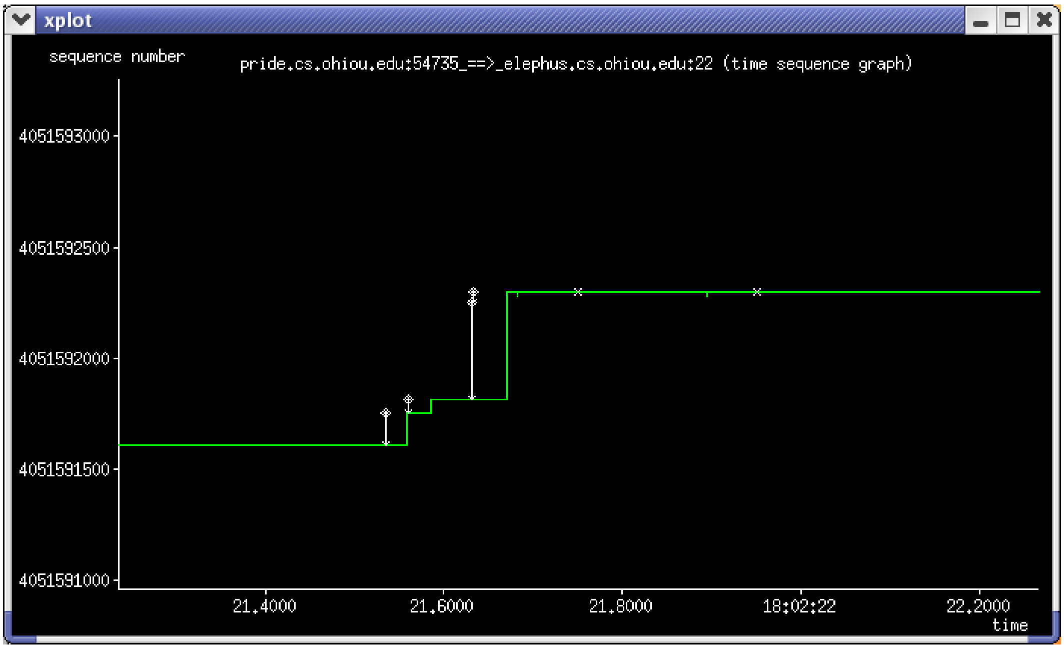


Figure B.6: PUSH segments. (Source: TCPTRACE Manual.)

URGENT segments, i.e., TCP segments carrying URGENT data with the URG flag set in the TCP header are represented with a red **U** on top of the segment. This is shown in Figure B.7.

The following other symbols also occur in Time Sequence graphs :

- **O** represents packets received out of order.
- **HD** represent Hardware Duplicates. Hardware Duplicates correspond to link layer

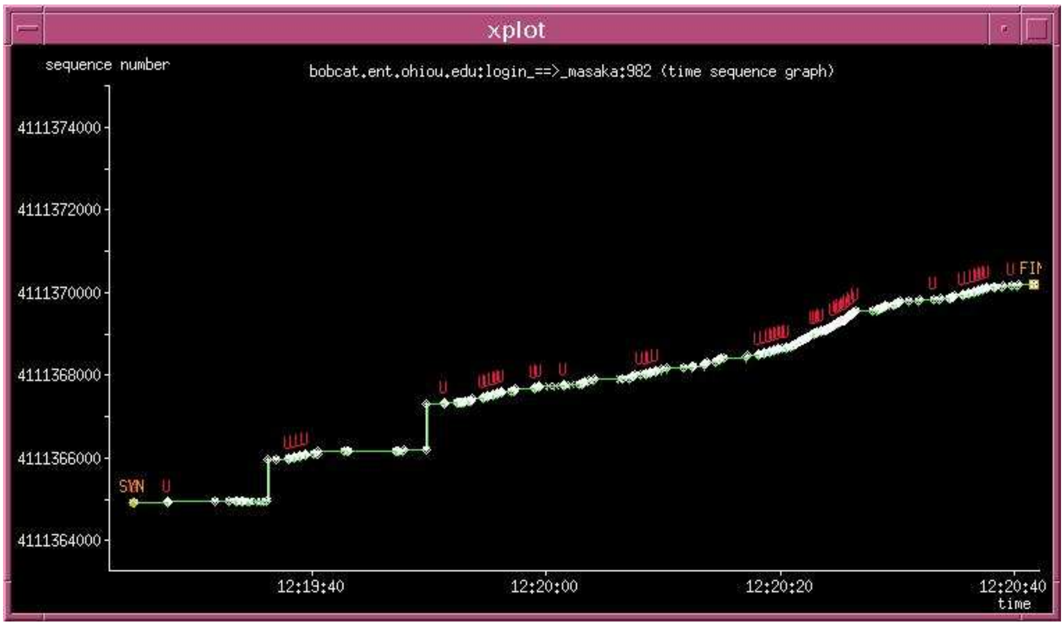


Figure B.7: URGENT segments. (Source: TCPTRACE Manual.)

retransmissions found when a duplicate packet with same IPv4 identification number and TCP sequence number as a previously observed packet is seen.

- **3** indicates that the received ack packet was the triple duplicate ack, commonly used as the threshold to trigger the TCP fast retransmit/recovery algorithm.
- **CWR / CE** track Explicit Congestion Notification [72] messages received. CWR indicates that the Congestion Window Reduced flag was set in the TCP header of the packet, while the CE flag indicates that the Congestion Experienced code-point was found in the IP header of the packet.

B.2.2 Throughput Graph

Throughput graphs (named X2Y_tput.xpl) are generated with the `-T` option. A sample throughput graph is shown in Figure B.8.

The graph has throughput in bytes/second on the Y-axis and time on the X-axis. The yellow dots can be turned off with the `-y` option. The red line can also be turned off by modifying “tcptrace” source code.

- **Yellow Dots** represent instantaneous throughput, defined as the size of the segment seen divided by the time since the last segment was seen (in this direction).
- **Blue Line** tracks the average throughput of the connection up to that point in the life time of the connection (total bytes seen / total seconds so far).

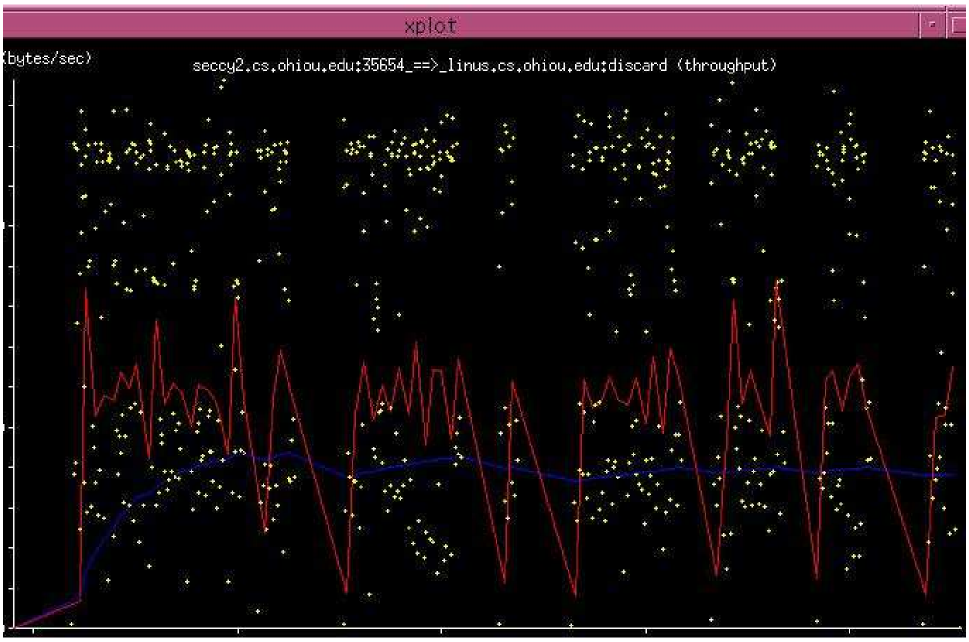

xplot
(bytes/sec)
seccy2.cs.ohiou.edu:35654_==>_linus.cs.ohiou.edu:discard (throughput)

Figure B.8: Throughput Graph. (Source: TCPTRACE Manual.)

- **Red Line** tracks the throughput seen from the last few samples, calculated as the average of N previous yellow dots. By default the line tracks the past 10 samples (N=10). However it can be changed with the `-AN` option. For example giving `-A5` along with the `-T` option calculates the throughput from the past 5 yellow dots to draw the line.