Optimization of Transmission Control Protocol and Feedback Control Mechanisms for Wireless Internet

A thesis submitted for the Degree of Master of Science in Computing (Thesis)

by

Phuong N. Nhan

Faculty of Information Technology

July, 2004

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis had not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis

Signature of Candidate

Production Note:
Signature removed prior to publication.

Preface

This research is funded by SPIRT (Strategic Partnership with Industry – Research and Training) Scheme in conjunction with the Motorola Australian Research Centre. The objective of the research project is to explore, propose, design and implement several algorithms that are applicable to the wireless networks in order to solve outstanding problems.

Firstly, the research investigates the relationship between packet loss and network congestion and introduces a feedback based end-to-end congestion control algorithm to the wireless network. The next algorithm is the new design of Explicit Loss Notification (ELN) at Base station in Wired-Cum-Wireless networks. With the combination of new ELN algorithm and Wireless FICC algorithm, the end-to-end performance and fairness are greatly improved by eliminating the misinterpretation of error related lost packets from congestion. Finally, the research investigates the effects of network congestion, which often happens over low bandwidth wireless link, and QoS performance (e.g. fairness, delay variation) of multiple sessions of TCP traffic in a hybrid network. We propose a framework, which consists of two main algorithms, feedback based congestion control and Explicit Window Adaptation (EWA).

Abstract

All current versions of reliable Transmission Control Protocol (TCP) react to packet losses differently and adjust the TCP congestion window in various ways. These protocols assume congestion in the network to be the primary cause for packet losses and unusual delays. TCP performs well over wired networks by adapting to end-to-end delays and packet losses caused by congestion. The TCP sender uses the cumulative acknowledgements it receives to determine which packets have reached the receiver, and provides reliability by retransmitting lost packets. The sender identifies the loss of a packet either by the arrival of several duplicate cumulative acknowledgements (say, three ACKs) or the absence of an acknowledgement for the packet within a timeout. TCP reacts to packet losses by reducing its transmission (congestion) window size before retransmitting packets, initiating congestion window or avoidance mechanisms and backing off its retransmission timer. These measures result in a reduction in the load on the intermediate links, thereby controlling the congestion in the network. Unfortunately, when packets are lost in the networks for reasons other than congestion, these measures result in an unnecessary reduction in end-to-end throughput and sub-optimal performance.

Wireless links typically have much higher bit error rates. This implies that packet loss would occur frequently. If no error correction is attempted at lower layer, TCP will exercise its congestion control procedure unnecessarily and the throughput will be reduced significantly. If the link layer performs error control by performing the retransmission itself, packet transmission time will vary greatly, sometime even exceeding TCP retransmission time out and again TCP slow start will occur. In wireless networks, "packet loss" problem is also encountered during handover when a mobile device moves from the coverage of one cell to that of another. During the handover, if the mobile station decides to make a handover before the segments are transmitted over the air interface, it is likely that some TCP segments buffered in a base station may be forwarded to another base station. This results in excessive segment delay or loss.

Thus, there is a clear demand for methods that can suppress the problems caused by the wireless environment. Recently, several techniques have been developed to improve end-to-

end TCP performance over wireless links. They can be classified into three categories: end-to-end TCP, split TCP and link layer TCP. However, they have not addressed these problems successfully.

In this thesis, we propose, design and implement several algorithms that are applicable to the wireless networks in order to solve outstanding problems. Firstly, the research investigates the relationship between packet loss and network congestion and introduces a feedback based end-to-end congestion control algorithm to the wireless network. This algorithm is a modification of a Fair Intelligent Congestion Control (FICC) proposed in [19]. The innovation of the algorithm is to modify the original FICC in such a way that the queue lengths can be effectively controlled when it is jointly employed with TCP in the wireless network.

The next algorithm is the new design of Explicit Loss Notification (ELN) at base station in Wired-Cum-Wireless networks. With the combination of new ELN algorithm and Wireless FICC algorithm, the end-to-end performance and fairness are greatly improved by eliminating the misinterpretation of error related lost packets from congestion.

Finally, the research investigates the effects of network congestion, which often happens over low bandwidth wireless link, and QoS performance (e.g. fairness, delay variation) of multiple sessions of TCP traffic in a hybrid network. We propose a framework, which consists of two main algorithms, feedback based congestion control and Explicit Window Adaptation (EWA).

Acknowledgements

This research was started when I was at School of Information Technologies (formerly Basser Department of Computer Science), the University of Sydney. It was resumed when I moved to the Department of Computer Systems, Faculty of Information Technology University of Technology, Sydney (UTS).

Many people have helped me all the way during the development of this thesis. First and foremost, I would like to thank my main supervisor A/Prof. Doan Hoang who took me on as his student, provided a motivating, enthusiastic, and critical atmosphere during the many discussions we had. It was a great pleasure for me to conduct this thesis under his supervision.

I am grateful to my co-supervisor Dr Vinod Mirchandani at Motorola Australian Research Centre, Sydney. He gave me very supportive guidelines in order to achieve the target of the research. We had meetings and discussions on my research on regular basis at Motorola Australian Research Centre, Sydney since I started my research.

I benefited greatly from Dr Hong Zhou providing constructive comments during my thesis time as well as giving valuable detailed comments on simulation results analysis. She also gave me great help in C++ coding for ns-2 implementations and excellent suggestions on algorithm designs.

I was fortunate to be a member of the Advance Research on Networking (ARN) group. I would like to thank everyone in the group including Xiaomei, Qing, Ian, Chi, Bushar, Hanh and Joe who shared experiences and knowledge with me from my starting points; a special thank to Ming Li who helped me on gaining general networking knowledge as well as on Network Simulation (ns-2).

Last but not least, I owe a special debt of gratitude to my grandparents and parents. They have, more than anyone else, been the reason for me to get this far. They give me love, and emotions to overcome all difficulties in life.

To my grandparents

and my parents

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Terminology

ACK	*Acknowledgement*
ATM	*Asynchronous Transfer Mode*
BS	*Base Station*
CCK	*Complimentary code keying*
cwnd	*Congestion Window*
DCF	*Distributed Coordinator Function*
DiffServ	*Differentiated Services*
DPF	*Down Pressure Factor*
ECN	*Explicit Congestion Notification*
ELN	*Explicit Loss Notification*
EWA	*Explicit Window Adaptation*
FICC	*Fair Intelligent Congestion Control*
FTP	*File Transfer Protocol*
GHz	*Giga-Hertz*
GPLS	*General Packet Radio Service*
GPRS	*General Packet Radio Service*
GSM	*Global*
HTTP	*Hypertext Transfer Protocol*
IntServ	*Integrated Services*
IP	*Internet Protocol*
IPv6	*Internet Protocol version 6*
ISO	*International Standardization Organization*
LAN	*Local Area Network*
MH	*Mobile Host*
OFDM	*Orthogonal Frequency Division Multiplexing*
PCF	*Point Coordinator Function*
PDA	*Personal Digital Assistance*

PER	*Packet Error Rate*
QoS	*Quality of Service*
RF	*Radio Frequency*
SACK	*Selective Acknowledgement*
ssthresh	*Slow Start Threshold*
TCP	*Transmision Control Protocol*
TDMA	*Time Division Multiple Access*
TELNET	*Telecommunications Network*
TOS	*Type of Service*
UDP	*User Datagram Protocol*
URL	*Universal Resource Locator*
WAN	*Wide Area Network*
WFICC	*Wireless Fair Intelligent Congestion Control*
WLAN	*Wireless Local Area Network*
WWW	*World Wide Web*

Chapter 1

Introduction

1.1. Motivation

The explosion of the Internet has been phenomenal, it can be accessed anywhere in the world thanks to its usability and ease of access. The Internet alters profoundly the way people communicate, search for information, doing business or research.

Wireless network technology has induced a rapidly growing industry. It also presents many new challenges to research in communications and networking. The main characteristic of the wireless network technology is mobility in which users can travel randomly within the service area while exchanging information. The main challenges in wireless networking include the handling high physical errors rate inherent in wireless channels, the mechanisms for handling the hand-off and mobility. The physical error rate is very high compared to its counterpart in the wired network where applications can assume a reliable service underneath. Handoff is the movement of a mobile node from one cell to another, hence requiring switching to another base station to handle the call. If

this happens during a communication session, application will experience severe packet loss rate and delay.

The Transmission Control Protocol (TCP) is the de facto standard for reliable data transmission in the Internet today. TCP has been tuned to work well over traditional wired networks, but its performance over wireless networks is worse. This performance degradation results from several effects:

- High physical errors rate inherent in wireless channel compared to its counterpart in the wired network where applications can assume a reliable service underneath it;
- Hand-off is a movement of a mobile node from one cell to another, hence requiring switching to another base station to handle the call, application will experience severe packet loss and delay; and
- Small TCP transmission windows due to low wireless channel bandwidths. Low channel bandwidths are common in many wireless networks as well as on many Internet paths today. This, and the short connections common in many Web transfers, lead to TCP windows being very small.

As illustrated in figure 1.1, TCP adapts well to wired network congestion, but in wireless networks, the congestion window size is reduced and instable resulting throughput degradation.

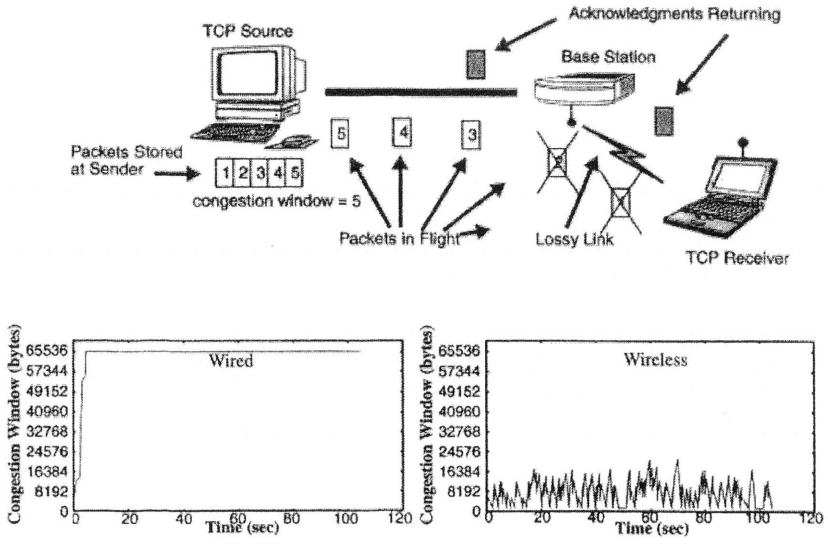


Figure 1.1 Comparison of TCP on wired and wireless networks [12]

In the past, several schemes have been proposed to alleviate the effects of non-congestion-related losses on TCP performance over networks that have wireless or similar high loss links. All schemes can be classified into three basic groups, based on their fundamental philosophy: *end-to-end proposals*, *split-connection proposals* and *link-layer proposals*. The end-to-end protocols attempt to make the TCP sender handle losses through the use of two techniques. First, they use some form of selective acknowledgements (SACKs) to allow the sender to recover from multiple packet losses in a window without resorting to a coarse timeout. Second, they attempt to have the sender to distinguish between congestion and other forms of losses using an Explicit Loss Notification (ELN) mechanism. At the other end of solution spectrum, split-connection approaches completely hide the wireless link from the sender by terminating the TCP connection at the base station. Such schemes use a separate reliable connection between the base station and the destination host. The second connection can use techniques such as negative or selective acknowledgements, rather then just regular TCP, to perform well over the wireless link. The third class of protocols, link-layer solution, lies between the other two classes. These protocols attempt to hide link-related losses from the TCP sender by using local retransmissions and perhaps forward error correction over wireless link. The local retransmissions use techniques that are tuned to the characteristics of the wireless link to provide a significant increase in performance. Since the end-to-end TCP connection passes the wireless link, the TCP sender may not be fully shielded from wireless losses. This can happen either because of timer interactions even for segments that are locally retransmitted. As a result, some proposals to improve TCP performance use mechanisms based on the knowledge of TCP messaging to shield the TCP sender more effectively and avoid competing and redundant retransmissions.

In this research, we address these challenges in detail and design solutions to them. These solutions incorporate link-layer techniques as well as enhancements to TCP at the sender and receiver.

The adverse effects of wireless bit-errors on TCP performance arise primarily because TCP misinterprets wireless losses as being due to congestion. The proposal is to develop a scheme similar to Fair Intelligent Congestion Control (FICC) protocol to serve as an end-to-end feedback loop. In addition, we show the development of the design for a mechanism called Explicit Loss Notification (ELN) that can successfully distinguish

between congestion and channel error-induced packet losses to substantially enhance end-to-end performance.

Another aim is to investigate the effects of network congestion, which often happens over low bandwidth wireless link, and Quality of Service (QoS) performance (e.g. fairness, delay variation) of multiple sessions of TCP traffic in a hybrid network. We proposed a framework, which consists of two main algorithms, feedback based congestion control and explicit window adaptation.

The other specific aim is to design and develop an optimized transport layer for wireless links. The transport layer may include QoS scheduling to help the operating system of and end system to deliver the bandwidth demanded by an application. The transport layer will integrate the QoS feedback loop control. Both of the ideas are illustrated in Figure 1.2.

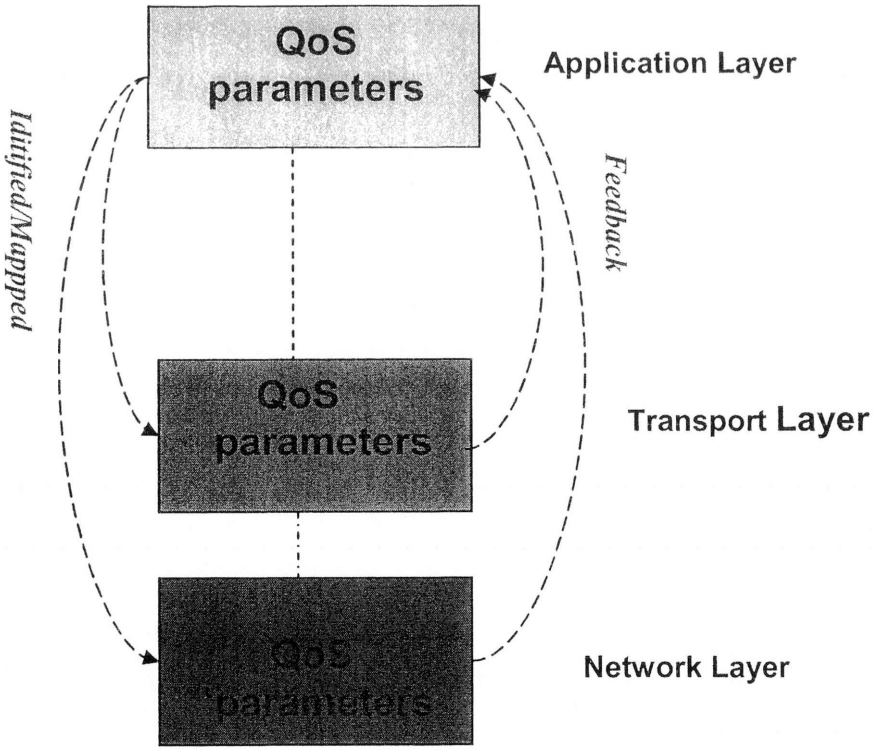


Figure 1.2 Overall research mapping layers

1.2 Contributions of the Thesis

In this thesis, we analyze the problems posed by the above challenges and design, implement and evaluate solutions to them that result in significant improvement in end-to-end performance. Because there is tremendous diversity in wireless access networks, and because there is enormous variation in their characteristics, there is no single panacea that cures all the observed problems. We recognize this and develop a suite of solutions and techniques to combat these problems that together comprehensively solve these problems. The components of this solution suite include the following algorithms:

1.2.1 Wireless Fair Intelligent Congestion Control (WFICC)

Transmission Control Protocol (TCP) has been deployed widely in the majority of commercial networks and services. It is thus essential for any wireless networking device that wishes to access these networks and their services to communicate via TCP/IP. Developed some two decades ago, TCP is based on assumptions that hold in wired networks [3], but not necessarily in wireless environment. TCP assumes that packet loss occurs primarily due to congestion somewhere in the network. This assumption obviously does not hold in unreliable high error-rate wireless networks. TCP interprets all packet losses are caused by congestions and it triggers the congestion slow start procedure unnecessarily. As a result, TCP performance is nowhere near as efficient as in wired networks.

There are various attempts in solving the problems by researchers [4,12] and can be categorized in three groups: *end-to-end proposals*, *split-connection proposals,* and *link-layer proposals.* . The end-to-end proposals keep a complete TCP connection from source to destination as in a wired network. This approach includes TCP SACK (Selective ACKnowledgement) and Explicit Loss Notification (ELN). TCP SACK can retransmit all known lost packets efficiently. ELN can provide the TCP sender the reasons for packet loss, but there have not been efficient solutions to implement it yet. The split-connection proposals such as Indirect TCP [15] decompose the TCP connection into two sub-connections for the wired and wireless parts of the path. However, this approach does not ensure end-to-end delivery of packets and does not work well if the connection is split

several times over the path of the connection. The link-layer proposals combine the ideas of the above two proposals. These protocols remain a complete end-to-end connection but distinguish from the end-to-end proposal by hiding link-loss from the TCP sender by using local error recovery. The examples are Snoop [16] and TULIP [17]. Snoop protocol introduces a snoop agent at the base station. The agent monitors every packet that passes in both direction and maintains a cache of TCP segments that have not yet been acknowledged by the receiver. It detects packet loss by duplicate acknowledgements or a local Retransmission TimeOut (RTO) and retransmits the lost packet. However, Snoop protocol does not work well in certain wireless environments where a fair RTO is difficult to calculate [18].

This algorithm isolates packet loss due to congestion and introduces a Window-based Wireless Fair Intelligent Congestion Control (WFICC) scheme to improve TCP performance. Simulations results demonstrate that by eliminating buffer overflows caused by congestions, WFICC provides better goodput, improves the delay and fairness of TCP connections significantly.

1.2.2 Wireless Fair Intelligent Congestion Control with Explicit Loss Notification (WFICC+ELN)

WFICC protocol has been designed by the Advanced Research in Networking (ARN) group to provide goodput improvement and fairness in wired-cum-wireless networks as described in 1.2.1. One disadvantage of WFICC is that the TCP source still does not know whether the end to end losses due to congestion or errors.

TCP performance in many wireless networks suffers because of packet losses induced by wireless bit errors, which occur in burst because of the nature of the wireless channel. Unfortunately, TCP wrongly attributes these losses to network congestion because of the implicit assumptions made by its congestion algorithms today. This causes the TCP sender to reduce its transmission window in response and often causes long timeouts during loss recovery that keep the connection idle for a long period of time. The result is degraded end-to-end performance. In addition, packet losses that occur due to user mobility cause the TCP sender to remain idle for long periods of times even after the handoff is completed, resulting in unacceptably low throughput [25].

A new Explicit Loss Notification (ELN) algorithm is designed at Base Station (BS). On the forward path, BS detects congestion related losses by examining out-of-order arriving packets and stores in a Hole list. On the backward path, when the BS receives duplicate ACK, it checks sequence number of an ACK to see if there exists any correspondence in the hole list, if not ELN flag in TCP header of the ACK is set to 1, so that the sources realize error-related losses and does not reduce the congestion window. The TCP sender retransmits the lost packet without reducing the window resulting in better performance.

Wireless Fair Intelligent Congestion Control with ELN (WFICC-ELN) is a combination of WFICC and ELN algorithms which give better performance and fairer goodput compared to its counterparts TCP and WFICC in the wired-cum-wireless networks.

In this part, we combine WFICC with ELN algorithms which give extra information on top of WFICC by setting ELN flag in the TCP ACK header so that the sender can distinguish between the lost packets which occur due to either congestion or error rate and decides whether to reduce the congestion window or leave it alone.

With combination of WFICC and ELN algorithms in the wired-cum-wireless networks, the performance is effectively improved, especially in a highly congested and high BER environment; the fairness of bandwidth is also successfully achieved.

1.2.3 Feedback-Based Congestion Control and Explicit Window Adaptation in A Hybrid Network with Wired and Wireless Links

It is well known that TCP performs poorly over wireless links as mentioned in 1.2.1 and 1.2.2 and this issue has attracted significant research interests. However, most of the research work emphasis on improving the throughput of TCP in a situation in which assumes that there is one session and the network is congestion free. This algorithm tries to investigate the effects of network congestion, which often happens over low bandwidth wireless link, and QoS performance (e.g. fairness, delay variation) of multiple sessions of TCP traffic in a hybrid network. We proposed a framework consisting of two main algorithms, namely *feedback based congestion control* and *explicit window adaptation.* Simulations results demonstrate that by eliminating buffer overflows caused by congestions, the framework provides desirable fairness, reduces delay variation and improves effective throughput.

The demand for providing Internet services over hybrid networks with wired and wireless links emerged in the recent past and will continue to grow rapidly. Transmission Control Protocol (TCP), one of the main protocols in TCP/IP stack, plays a key role in providing reliable data services in the Internet. However, TCP was initially designed for high quality wired networks about 20 years ago. Although there are few TCP variants developed in succession (e.g. Tahoe, Reno, NewReno, SACK), TCP cannot react efficiently to high error rate in wireless environments.

Enhancements for TCP to deal with high error rate can be done on two aspects. 1). Decouple the congestion control from retransmission of lost packets. In regular TCP, whenever packet loss is detected, TCP sender retransmits the packet and shrinks its window size at the same time as TCP assumes that all packet losses are due to congestions somewhere in the network and congestion in TCP is not acknowledged until packet loss or timeout occurs. 2). Detect and retransmit the lost packet as early as possible. Retransmission of the lost packet locally improves the efficiency of TCP. Link layer protocols such as Forward Error Correction (FEC) and Automatic Repeat reQuest (ARQ) make the wireless link more reliable.

Following the idea of above two ways, one or the other, many schemes have been proposed to address the issue. The representative examples are Explicit Loss Notification (ELN), Explicit Congestion Notification (ECN), Indirect TCP, and Snoop. ELN can provide the TCP sender the reasons of packet loss, but there have been no efficient solutions to implement it yet. Indirect TCP decompose the TCP connection into two sub-connections for the wired and wireless parts of the path. However, this approach does not ensure end-to-end delivery of packets and does not work well if the connection is split several times over the path of the connection

Different from the previous research work, the research work presented in this part tries to investigate the effects of network congestion and solve the deficiency of TCP in misinterpreting the link-related packet loss. It is focused on improving Quality of Service (QoS) performance of TCP for multiple sessions in a hybrid network. A framework is proposed which consists of the following two components:

1) Feedback congestion control. We use Fair Intelligent Congestion Control (FICC) algorithm [1] as the feedback congestion control algorithm. FICC continuously collects network traffic information and, based on the information, calculates the fair share for individual sessions. The fair share is continuously updated and feed backed to the Source Edge Router (SER).

2) Traffic shaping. A traffic shaper at the SER sends the packets from individual sessions at the rate of fair share. An explicit window adaptation (EWA) algorithm calculates explicit window size (EWS) based on the shaping buffer fill level. The EWS is continuously updated and feed backed to the TCP sender by an ACK packet in its header as receiver window size.

We implement the framework in Network Simulator Version 2 (NS2) [2]. The simulation results show that the framework can successfully avoid congestion(s), achieve desirable fairness, reduce delay variations and improve the effective throughput in a hybrid network.

1.3 Structure of the Thesis

The rest of this thesis is organized as follows. In chapter 2, we discuss background material in the area of reliable transport protocols and describe the related work in improving transport performance over wireless networks. In Chapter 3, we describe the details of our research methodology, which include Network Simulation 2 (ns-2) and Fair Intelligent Congestion Control Algorithm.

Chapters 4 through 6 form the core of the thesis. Chapter 4 describes the Fair Intelligent Congestion Control in Wired-Cum-Wireless Networks (WFICC), designed to isolate packet loss due to congestion and introduce a Window-based Fair Intelligent Congestion Control (WFICC) scheme to improve TCP performance.

Chapter 5 combines newly designed Explicit Loss Notification (ELN) algorithm with previous Fair Intelligent Congestion Control in Wired-Cum-Wireless Networks (WFICC) which even gives better TCP performance in throughput and fairness.

Chapter 6 investigates the effects of network congestion, which often happens over a low bandwidth wireless link, and QoS performance (e.g. fairness, delay variation) of multiple sessions of TCP traffic in a hybrid network. We propose a framework, which consists of two main algorithms, feedback based congestion control and explicit window adaptation.

Finally, in Chapter 7, we present a summary of our work and contributions. We conclude this thesis with a discussion of general lessons learned about adaptive protocol design for heterogeneous networks and outline some directions for future research.

Chapter 2

Background and Related Work

The purpose of the chapter is to introduce the background and related work in reliable transport protocols and reliable data delivery over wireless networks.

2.1 The Internet Protocol Architecture

The Internet is a best-effort network. Today's Internet infrastructure provides no guarantees on bandwidth or latencies and has no mechanisms to avoid packet losses. Packets between end-hosts are routed through a set of routers. Typically, in conventional wired networks, network congestion due to overload and router resource contention causes packet loss. This service model leads to a simple internal architecture that scales to handle a large number of communicating entities, but requires higher level protocols and applications to adapt to packet loss, varying available bandwidths and latencies.

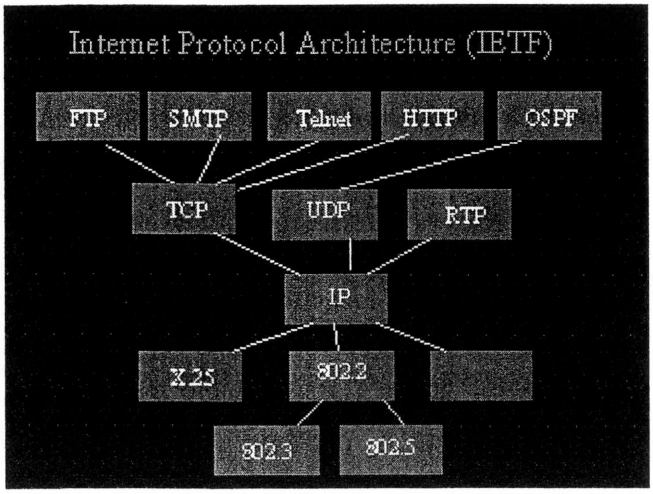


Figure 2.1 Internet Protocol Architecture (EITF)
(http://www.soi.wide.ad.jp/class/99007/slides/03/13.html)

2.1.1 Heterogeneity

The Internet is characterized by an enormous degree of heterogeneity, End-hosts range from powerful servers and desktop computers to so-called "thin-clients" and hand-held devices; network links range from multi-gigabit wired links to low-bandwidth dialup and wireless links; protocol range from unreliable unicast protocol (e.g., UDP [40]) to reliable multicast protocols [41]; and applications range from bandwidth-sensitive video conferencing (e.g., [42]) to latency-sensitive remote terminal applications [e.g., [43]]. Despite this diversity, the Internet enables hosts to communicate with each other independent of the specific characteristics of the machines or access technology. This is achieved by using the Internet Protocol (IP) [44], which provides a common vocabulary for communication between hosts. That is, end-hosts can communicate with one another as long as they use IP for addressing and communication, via a network of routers that run IP. Thus, IP enables universal connectivity independent of host and link technology, making it the *lingua franca* of the Internet today.

2.1.2 Reliability

By bringing together a range of disparate technologies, IP solves one aspect of the heterogeneity problem: that of enabling basic connectivity between hosts. However, because the IP service model provides the abstraction of a best-effort network, packet losses can and do happen. There are numerous applications, such as World Wide Web access (e.g. Using HTTP [45, 46]), file transfers (e.g., ftp [48]), electronic mail [47] interactive logins (e.g., telnet [43]), etc. that require reliable, ordered data delivery from the network. This is provided by a reliable transport protocol that runs over IP that such applications can use via a well-defined socket interface [49]

To work well the global Internet, a good reliable transport protocol must perform two critical functions: loss recovery and congestion control. Loss Recovery is the ability of the sender to deduce that certain packets did not reach the receiver and retransmit them, either when requested by the receiver or without any explicit request from the receiver. Congestion control is the ability of the sender to recognize situations when the network is overloaded and react to it by reducing its transmission rate (window size).

Thus, TCP congestion control and loss recovery mechanisms are tightly coupled to each other, which lead to degraded performance in wireless networks where losses frequently occur for reason other than congestion. Furthermore, any irregularity in the ACK stream skews TCP's ACK-clocking mechanism and manifests itself in the data stream in the immediate future. This situation arises in bandwidth-asymmetric and packet radio networks.

2.1.3 Incremental Deployment

We use TCP/IP protocol architecture as the starting point for our solutions to wireless problems rather than designing a new protocol architecture completely from scratch. There are significant advantages in adopting this approach. The first is the ability to incrementally deploy our changes in the global Internet and realize the vision of making wireless devices first-class citizens in the Internet. The second is being able to operate with currently-deployed legacy protocol implementations at certain hosts in the network, obviating the need

for a "flag day", a day on which all Internet hosts need to change to a new protocol architecture.

According to previous works and our recent researches, using combination of TCP options, local-link layer solutions, modifications and enhancements to TCP's algorithms, especially out Wireless Fair Intelligent Congestion Control combined with Explicit Loss Notification algorithm, we have been able to comprehensively solve many observed problems to efficient wireless data transport. Our research demonstrates that flexibility of the TCP/IP protocol architecture and strongly discourages the need to design a new protocol framework to combat the problems observed in the heterogeneous networks investigated in this thesis. Thus, our solutions enable a variety of wireless technologies to be successfully and efficiently integrated into the Internet in an incremental fashion.

2.2 Transmission Control Protocol (TCP)

TCP is the de factor standard for reliable data transport in the Internet today. While the original formal specification of TCP is in RFC 793, numerous variants of it have been developed over the past several years, such as Tahoe, Reno, Vegas, etc..

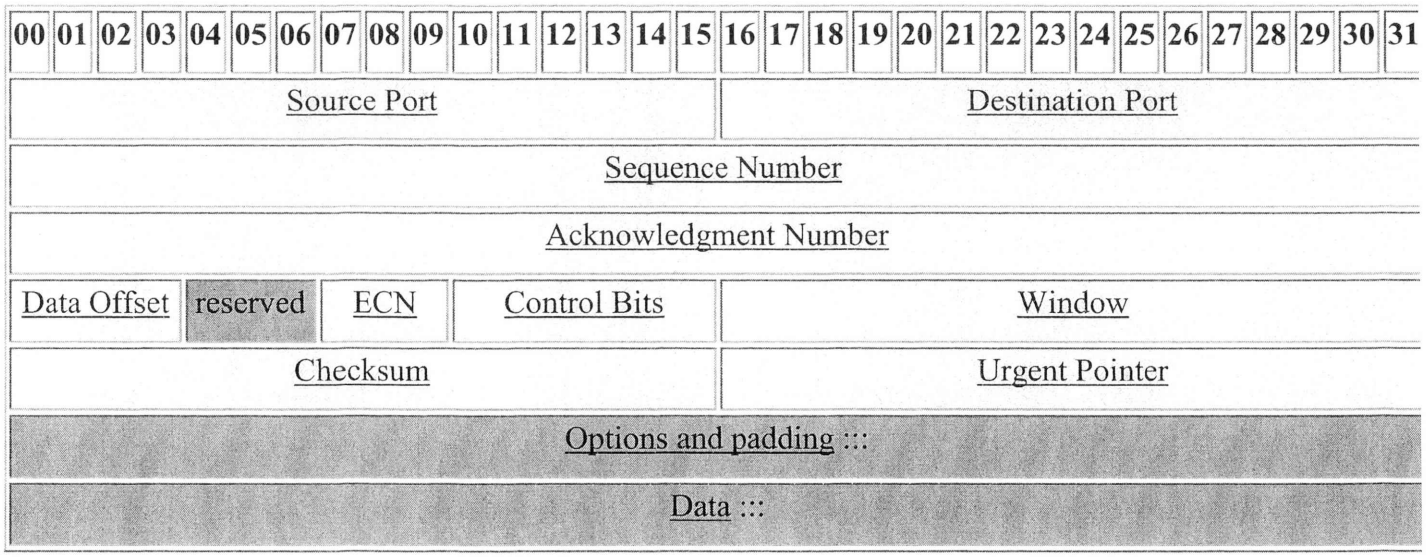


Figure 2.2 TCP Header

2.2.1. Cumulative Acknowledgments

TCP is an ARQ-based reliable transport protocol that uses cumulative ACKs and byte-based sequence numbers for reliability. TCP provides a fully reliable, in-order, byte-stream delivery abstraction to the higher-layer application, which typically uses a socket interface [49] to interface with the transport layer. The basic unit of transmission is called a segment, which is a contiguous sequence of bytes identified by its 32-bit long start and end sequence numbers. The transmitted segments are smaller than or equal to the connection's maximum segment size (MMS), which is negotiated at the start of the connection.

2.2.2. Loss Recovery

When the TCP sender discovers that data has been lost in the network, it recovers from it by retransmitting the missing segments. TCP has two mechanisms for discovering and recovering from losses: time-driven retransmissions and data-driven retransmissions.

Time-driven recovery: When the TCP sender does not receive a positive cumulative ACK for a segment within a certain timeout interval, it retransmits the missing data. To determine the timeout interval, it maintains a running estimate of the connection's round-trip time using an exceptional weighted moving average (EWMA) formula, srtt = a*rtt + (1-a)*srtt, where srtt is the smoothed round-trip time average, rtt the current round-trip sample, and a the EWMA constant set to 0.125 in the TCP specification. It also estimates the mean linear deviation, rttvar, using a similar EWMA filter, with a set to 0.25. A timeout occurs if the sender does not receive an ACK for segment within srtt+4*rttvar since the arrival of the last new cumulative ACK. Furthermore, the transmission timer is exponential backed off after each unsuccessful retransmission. The details of the roundtrip time calculations and timer management can be found in [9, 46, 50].

Data-driven recover: TCP's data-driven retransmission mechanism uses a technique called Fast-Retransmission. It relies on the information conveyed by cumulative ACKs and takes

advantage of the receipt of later data segments after a lost one. Because ACKs are cumulative, all segments after a missing one generate duplicate cumulative ACKs that are sent to the TCP sender. The sender uses these duplicate cumulative ACKs to deduce that a segment is missing and retransmits it.

However, the sender must not retransmit a segment upon the arrival of the very first duplicate ACK. This is because the Internet service model does not preclude the reordering of packets in the network; such reordering causes later segments to be received ahead of earlier ones, and triggers duplicate ACKs in the same way that losses do. Furthermore, the degree of packet reordering on the Internet seems to be increasing, according to studies of Paxson [52]. Thus, to avoid prematurely retransmitting segments, the sender waits for three duplicate ACKs, the current standard fast retransmit threshold.

This is followed by the fast recovery phase, where additional packets are transmitted after the sender is sure that at least half the current window has reached the receiver, based on a count of the number of received duplicate ACKs. Fast recovery ensures that a fast retransmission if followed by congestion avoidance and not by slow start. Since the arrival of duplicate ACKs signals to the sender that data is indeed flowing between the two ends, there is no reason to suddenly throttle the sender by invoking slow start.

For retransmissions are often not sufficient to recover from multiple losses in a single window, because all the cumulative duplicate ACKs arrive for the first loss in the window. This usually results in a coarse-grained timeout before the packet is retransmitted.

TCP currently makes the tacit assumption that all losses occur because of network congestion. Thus, coupled with either of these modes of retransmission is congestion control that reduces the sender's transmission rate. The next section discusses how TCP manages congestion by probing sustainable bandwidth and reacting to congestion.

2.2.3. Congestion Avoidance and Control

TCP's congestion management is based largely on Jacobson's seminal paper [9] and on the principles of multiplicative-decrease/additive-increase expounded by Chiu and Jian [20]. TCP uses a window-based algorithm to manage congestion, where the window is an estimate of the number of bytes currently unacknowledged and outstanding in the network.

TCP sender performs flow control by ensuring that the transmission window does not exceed the receiver's advertised window size. It performs congestion control by using a window-based scheme, where the sender regulates the amount of transmitted data using congestion window. When a connection starts or resumes after an idle period of time, slow start is performed. Here, there congestion window is initialized to one segment and every new ACK increases the window by one MSS. After a certain threshold (called the slow start threshold, *ssthresh*) is reached, the connection moves into the congestion avoidance phase, in which the congestion window effectively increases by one segment for each successfully transmitted window. In response to a packet loss, the sender halves its congestion window; if a timeout occurs, the congestion window is set to one segment and the connection goes through slow start once again.

In addition, *ssthresh* is set to half the value of *cwnd* at the time of loss, and at each stage slow start is performed until *cwnd* reaches ssthresh. At any point in time, the TCP sender ensures that the number of unacknowledged bytes is not larger than the smaller of the receiver's advertised flow control window and cwnd. In the steady state of the connection, the sender transmits a window's worth of data every round-trip, at a long-term rate equal to the bottleneck bandwidth. Since the ratio of window to round-trip delay equals the bottleneck bandwidth, TCP's congestion window size tends towards the connection's bandwidth-delay product. This is the number of outstanding bytes in transit in the steady state. Thus, it is clear that a low-bandwidth connection implies small transmission windows. In turn, this affects the fast retransmission mechanism because not enough duplicate ACKs arrive, leading to expensive timeouts.

Data transmissions are triggered and clocked by ACKs that arrive for previous segments; every time an ACK arrives, the transmission window shifts by an amount equal to the sum of the number of bytes acknowledged and the increase in cwnd, subject to the constrains imposed by the flow control window. This elegant notion of transmitting data based on incoming ACKs, called ACK-clocking, makes TCP a tightly-coupled feedback system and frees the sender from maintaining explicit software timers for transmission. However, it also leads to significant performance degradation when ACK feedback is imperfect or variable.

2.2.4. Connection Management

TCP has a sophisticated connection initiation mechanism using a three-way handshake, where a SYN exchange occurs before the connection is established and data can flow. A connection is uniquely identified by the source and destination IP addresses and port numbers.

2.2.5. Window-based vs. Rate-based Congestion Control

TCP's window-based congestion control algorithm is elegant because it does not need an explicit clock to pace its data transmission, since ACK clocking ensures that new data segments are sent out triggered by incoming ACKs. However, it is well-known that several problems arise from using the sender's transmission window to perform both congestion and error control in window-based protocol such as TCP. These arise because of the tacit assumption that all losses are the result of network congestion, and the use of the same window for both loss recovery and congestion management. The alternate approaches proposed by several authors attempt to achieve a separation of congestion control and loss recovery by performing congestion avoidance using a rate-based scheme. The basic idea is that the sender controls the rate at which packets are transmitted into the network using a timer. That is, the ACK-clocking mechanism used in TCP, is avoided in favor of a mechanism that is more explicit and not window-based.

2.3 End-to-end TCP Enhancements

Over the past decade, TCP has been tuned to work well in wired networks in the face of network congestion, reacting to congestion by reducing its rate, recovering from lost segments, and probing for bandwidth in a careful way. In this section, we survey some proposed enhancements to TCP that improve its ability from losses in a timely manner and/or perform better congestion control.

2.3.1. Selective Acknowledgement (SACK)

It is well known that TCP performance suffers due to coarse-grained timeouts when multiple segments are lost in a single window because it uses only cumulative ACKs. There has therefore been recent interest in adding selective acknowledgements (SACKs) to the standard TCP specification to reduce the time it takes to recover from multiple losses in a window. Fall and Floy describe detailed simulation results in highlighting the benefits of SACKs in large delay-bandwidth product networks.

TCP Selective Acknowledgement can be implemented in many ways. One possible approach is to use Keshav and Morgan's SMART (Selective Mechanism to Aid ReTransmission) scheme, where the receiver communicates the segment number that just arrived in addition to the cumulative ACK, whenever it sends a duplicate ACK. A second approach, described in RFC 2018, is currently an Internet standard. In this scheme, the receiver reports up to three of the last received, out-of-order, maximum contiguous blocks of data, in addition deduce which segments have reached the receiver. The information reported by the receiver in this scheme is a superset of the information reported in the SMART. By reporting up to three blocks of data, it ensures that every block is reported at least thrice to the sender; this makes it robust to the loss of some SACKs on the reverse path.

2.3.2. Newreno

Newreno modification to TCP Reno reduces the number of timeouts incurred by the TCP sender when multiple losses happen in a transmission window. When multiple losses occur in TCP Reno, a fast retransmission occurs after three duplicate ACKs arrive. Later duplicate ACKs are ignored until half the window is acknowledged, after which fast recovery sends a new segment for every incoming duplicate ACK. Now, when a new ACK arrives after the successful fast retransmission, its value will be within the original window (recall that there were multiple losses in the original window). In many cases TCP Reno would timeout for this second loss if the second loss is "close" to the location of the first one, because the sender's window would now have been reduced to half its original window to beyond the original right edge.

In Newreno, however, the sender remains in fast recovery when this happens, when the new ACK arriving after a fast retransmission is partial. A partial ACK is defined as a cumulative ACK that does not acknowledge the original window completely. By remaining in fast recovery, the sender continues to send new segments, which would elicit more duplicate ACKs and eventually trigger another fast retransmission without incurring a timeout.

2.3.3. Explicit Congestion Notification

When a gateway en route is congested or close to congestion, it sets a bit in the packet header and forwards it on. A Random Early Detection (RED) [26] gateway in marking mode is apt for this. A RED gateway detects incipient congestion by tracking the average queue size over a time window in the recent past. If the average exceeds the threshold, the gateway selects a packet at random and probabilistically marks it, by setting the Explicit Congestion Notification (ECN) bit in the IP packet header [24,31]. This notification is echoed to the sender receives and ACK with ECN, it multiplicatively reduces its congestion window, as it would if a packet loss had occurred. Thus, this mechanism with explicit support from gateway allows TCP to perform proactive congestion control, over and above the reactive one triggered by packet loss.

2.4.4. Explicit Loss Notification

TCP has been tuned for traditional networks comprising wired links and stationary hosts. It assumes *congestion* in the network to be the primary cause for packet losses and unusual delays, and adapts to it. The TCP receiver sends cumulative acknowledgments (ACKs) for successfully received segments, which the sender uses to determine which segments have been successfully received. The sender identifies the loss of a packet either by the arrival of several duplicate cumulative ACKs, triggering a fast retransmission, or by the absence of an ACK for a timeout interval equal to the sum of the smoothed round-trip delay and four times its mean deviation. TCP reacts to packet losses by retransmitting missing data, and simultaneously invoking congestion control by reducing its transmission (congestion) window size and backing off its retransmission timer. These measures reduce the level of congestion on the intermediate links.

Unfortunately, when packets are lost for reasons other than congestion, these measures result in an unnecessary reduction in end-to-end throughput and hence, in sub-optimal performance. Communication over wireless links is often characterized by high bit-error rates due to channel fading, noise or interference, and intermittent connectivity due to handoffs. TCP performance in such networks suffers from significant throughput degradation and very high interactive delays because the sender misinterprets corruption for congestion.

Explicit Loss Notification (ELN) is used to inform the sender that the packet losses due to error but not congestion so that it does not reduce the congestion window. This improves throughput significantly.

2.4 Wireless Networks

2.4.1 Wireless Technology Overview

Wireless Computing is a rapid emerging technology providing users with network connectivity without being tethered off of the wired network. Ideally users of wireless networks want the same services and capabilities that they have commonly come to expect

with wired networks. However, to meet these objectives, the wireless community faces certain challenges and constrains that are not imposed on their wired counterparts.

There are two common kinds of spread spectrum technologies: direct-sequence and frequency-hopping. In direct-sequence spread spectrum, the input signal is transmitted simultaneously over a broad range according to a pre-assigned code. In frequency-hopping spread spectrum, the transmitter sends data on one narrowband frequency for a short amount of time, and then jumps to another narrowband frequency, using a pseudo-random hopping sequence. The receiver synchronizes with the sender's hopping sequence and tunes in to those frequencies to receive and decode the signal.

While frequency-hopping is more robust to sources of interference than spread-spectrum, maximum data rates are typically lower. Thus, it is not as suitable for high-speed wireless LAN access as for slower, wide-area links.

2.4.2 IEEE 802.11 Concepts

The IEEE is developing an international WLAN standard identified as IEEE 802.11 [37]. This project was initiated in 1990, and several drafts have been published for review. The scope of the standard is "to develop a Medium Access Control (MAC) and Physical Layer (PHY) specification for wireless connectivity for fixed, portable and moving stations within the local area." The purpose of the standard is twofold:

- "To provide wireless connectivity to automatic machinery, equipment, or stations that require rapid deployment, which may be portable, or hand-held or which may be mounted on moving vehicles within a local area";
- "To offer a standard for use by regulatory bodies to standardize access to one or more frequency bands for the purpose of local area communication".

There are two different modes of operation for IEEE 802.11 devices: *ad hoc* (Independent Basic Service Set, IBSS) *or infrastructure* (Extended Service Set, ESS). An ad hoc network

is usually one that exists for a limited time between two or more wireless devices that *is not* connected through an access point (AP) to a wired network.

Infrastructure mode assumes the presence of one or more APs bridging the wireless media to the wired media. The AP handles station authentication and association to the wireless network. Multiple APs connected by a distribution system (DS) can extend the range of the wireless network to a much larger area than can be covered by any one AP. In typical installations, the DS is simply the existing IP network infrastructure. For security purposes, virtual LANs (VLANs) are often used to segregate wireless traffic from other traffic on the DS. Although 802.11 allows for wireless stations to dynamically switch association from one access point to another (as would be the case of a mobile PDA user walking across a campus), it does not govern how this is to be accomplished. As a result, vendor implementations are generally not interoperable in this respect. At the time of this writing, implementing this type of functionality requires a single vendor solution.

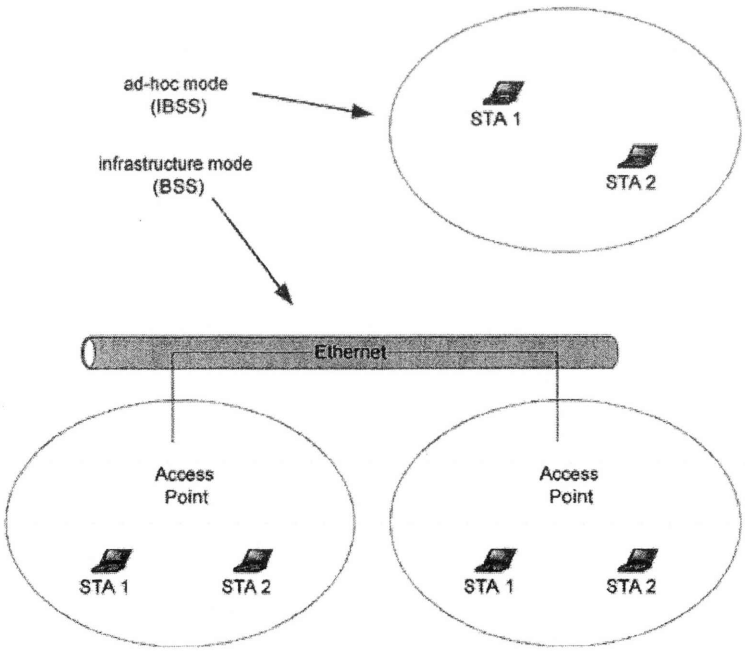


Figure 2.3 Ad hoc mode vs. infrastructure mode

The products - based on the 802.11b standard - are faster, lower in cost, and simpler to setup and use than previous generation products. The majority of WLAN products today

communicate at speeds up to 11 megabits per second (Mbps).

Two new WLAN standards are now emerging and will deliver higher speeds, up to 54 Mbps, to WLAN users. These new standards are known as 802.11a and 802.11g. To help you to better understand these new technologies, the Wireless LAN Association (WLANA) has developed this white paper as an introduction to high-speed WLAN options.

The first wireless LAN standard, 802.11, was approved by the Institute of Electrical and Electronics Engineers (IEEE) in 1997 and supported speeds up to 2 Mbps. In 1999, the IEEE approved both the 802.11a and 802.11b standards. 802.11a specified radios transmitting at 5 GHz and at speeds up to 54 Mbps using orthogonal frequency division multiplexing (OFDM) modulation technology. The 802.11b standard - now popularly known as Wi-Fi - specified operation in the 2.4 GHz band (also known as the ISM band) and could achieve speeds up to 11 Mbps using direct sequence spread spectrum (DSSS) technology.

Because DSSS is easier to implement than OFDM, 802.11b products appeared on the market first, starting in late 1999. Since then, 802.11b products have been widely deployed in corporations, small offices/home offices (SOHO), in residential home and in public locations (Wi-Fi "hotspots"). Products bearing the Wi-Fi logo conform to the 802.11b standard, have passed an interoperability certification test defined by the Wireless Ethernet Compatibility Alliance (WECA) and have received permission from WECA to use the logo. In early 2002, the first end-users products based on the 802.11a standard were shipped.

The 802.11g standard brings the benefits of higher speeds, while maintaining backward compatibility with existing 802.11b equipment. 802.11g specifies operation in the same 2.4 GHz frequency band and with the same DSSS modulation types as 802.11b at speeds up to 11 Mbps, while adding more efficient OFDM modulation types at higher speeds.

An 802.11g network card, for example, will work with an 802.11b access point and an 802.11g access point will work with 802.11b network cards - at speeds up to 11 Mbps. To benefit from higher speeds up to 54 Mbps, both the access point and network card must be

802.11g compliant. The draft standard also specifies optional modulation types (OFDM/CCK) that are intended to improve efficiency in an all-802.11g installation.
The tradeoff with 802.11g is in a lower capacity, versus 802.11a, to serve a large number of high- speed WLAN users. The OFDM modulations allow for higher speed but the total available bandwidth in the 2.4 GHz frequency band remains the same because 802.11g is still restricted to three channels in the 2.4 GHz band, unlike the eight that are available in the 5 GHz band.

Of particular interest in the specification is the support for two fundamentally different MAC schemes to transport asynchronous and time-bounded services. The first scheme, distributed coordination function (DCF), is similar to traditional legacy packet networks supporting best-effort delivery of the data. The DCF is designed for asynchronous data transport, where all users with data to transmit have an equally fair chance of accessing the network. The point coordination function (PCF) is the second MAC scheme. The PCF is based on polling that is controlled by an access point (AP). The PCF is primarily designed for the transmission of delay-sensitive traffic. While the DCF has been studied by several researchers, the combined performance of the DCF and PCF operating in a common repetition interval is much less understood.

2.5. Wireless Bit-Errors and User Mobility

2.5.1. Reasons for Wireless Data Corruption

Recently, several schemes have been proposed to the alleviate the effects of non-congestion-related losses on TCP performance over networks that have wireless or similar high loss links. These schemes choose from a variety of mechanisms, such as local transmission, split-TCP connections, and forward error correction, to improve end-to-end throughput. However, it is unclear to what extent each of the mechanisms contributes to the improvement in performance

The schemes are classified into three basic groups, based on their fundamental philosophy: end-to-end proposals, split-connection proposal and link-layer proposals. The *end-to-end protocols* attempt to make the TCP sender handle losses through the use of two techniques. First, the use some form of selective acknowledgements (SACKs) to allow the sender to recover from multiple packet losses in a window without resorting to a coarse timeout. Second, they attempt to have the sender distinguish between congestion and other forms of losses using an Explicit Loss Notification (ELN) mechanism.

2.5.2. End-to-End Protocols

The end-to-end protocols attempt to make the TCP sender handle losses through the use of two techniques. First, they use some form of selective acknowledgements (SACKs) to allow the sender to recover from multiple packet losses in a window without resorting to a coarse timeout. Selective acknowledgements are yet another way of dealing with the wireless loss issue, and are defined as an option to TCP in RFC 2108. Since standard TCP uses a cumulative acknowledgement scheme, it does not provide the sender with sufficient information to recover quickly when multiple packets are lost within a single transmission window. With the SACK scheme, each acknowledgement contains information up to three noncontiguous blocks of data that have been received successfully by the receiver. This allows TCP senders to retransmit only the missing data segments, the main drawback of SACK is that it requires modifying the acknowledgement procedure at the sender and the receiver. Second, they attempt to have the sender to distinguish between congestion and other forms of losses using an Explicit Loss Notification (ELN) mechanism. The TCP congestion control mechanisms are not invoked when the packet loss is due to a non-congestion related cause.

End-to-end schemes involve modifying the TCP implementation in the sender to make it "wireless" aware. This may not always be feasible and is a potential problem for widespread implementation of end-to-end schemes.

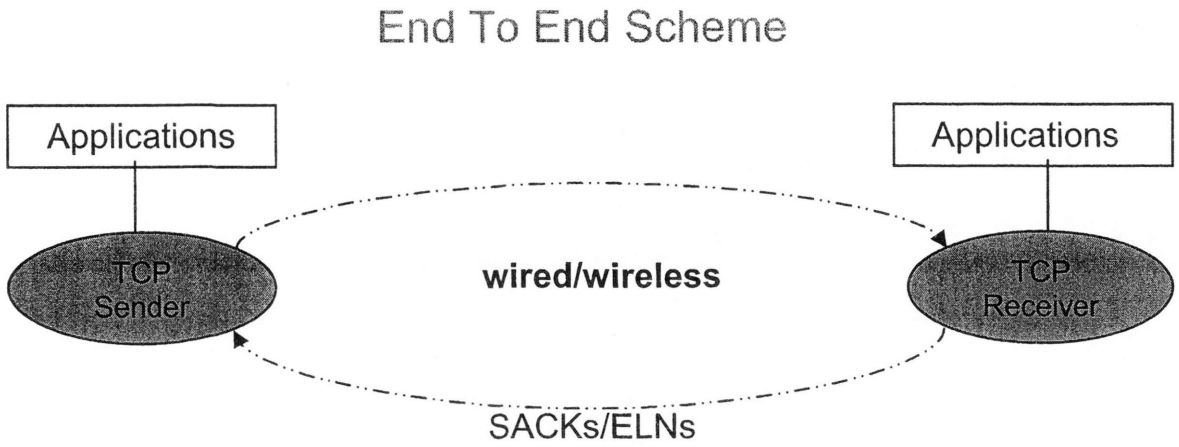


Figure 2.4 End-To-End Scheme

2.5.3. Split-Connection Protocols

In the mid-1990s, split-connection approaches gained in popularity as a way to improve TCP performance over error-prone wireless networks. They were motivated by the observation that the problems arose because the sender was imperfectly shielded from the error-prone link.

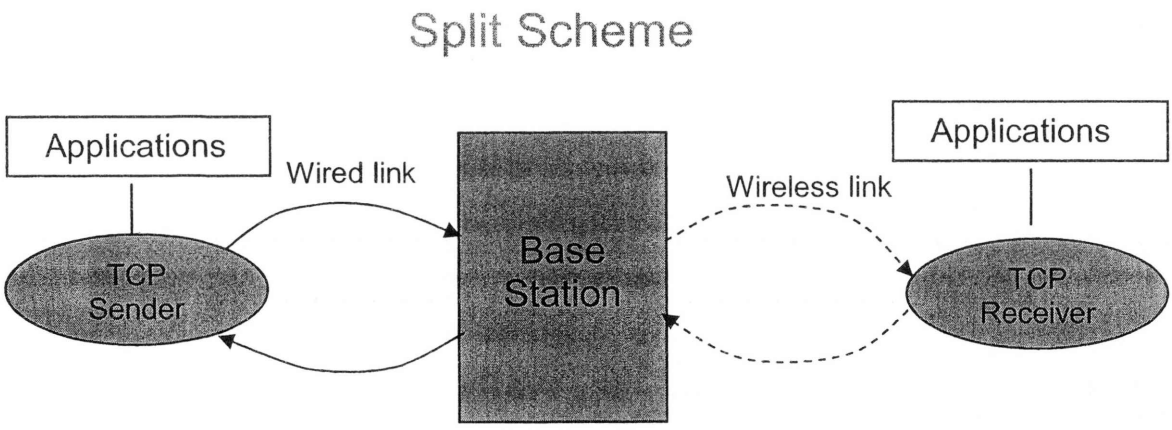


Figure 2.5 Split Scheme

To effect better shielding, split-connection protocols split each TCP connection between a sender and receiver into two separate connections at the base station – one TCP connection

between the sender and the base station, and the other between the base station and the receiver (Figure 2.5). Over the wireless hop, a specialized protocol tuned to the wireless environment may be used. Now, loss recovery for wireless losses can be done on the second connection without the data sender retransmitting data or reducing its window.

2.5.4 Link-layer Protocols

The third class of protocols, *link-layer solutions* lie between the other two classes. These protocols attempt to hide link-related losses from the TCP sender by using local retransmissions and perhaps forward error correction over wireless link. The local retransmissions use techniques that are tuned to the characteristics of the wireless ink to provide a significant increase in performance. Since the end-to-end TCP connection passes through the wireless link, the TCP sender may not be fully shielded from wireless losses. This can happen either because of timer interactions between the two layers, or more likely because TCP's duplicate acknowledgements causing sender fast retransmissions even before segments that are locally retransmitted. As a result, some proposals to improve TCP performance use mechanisms based on the knowledge of TCP messaging to shield the TCP sender more effectively and avoid competing and redundant retransmissions

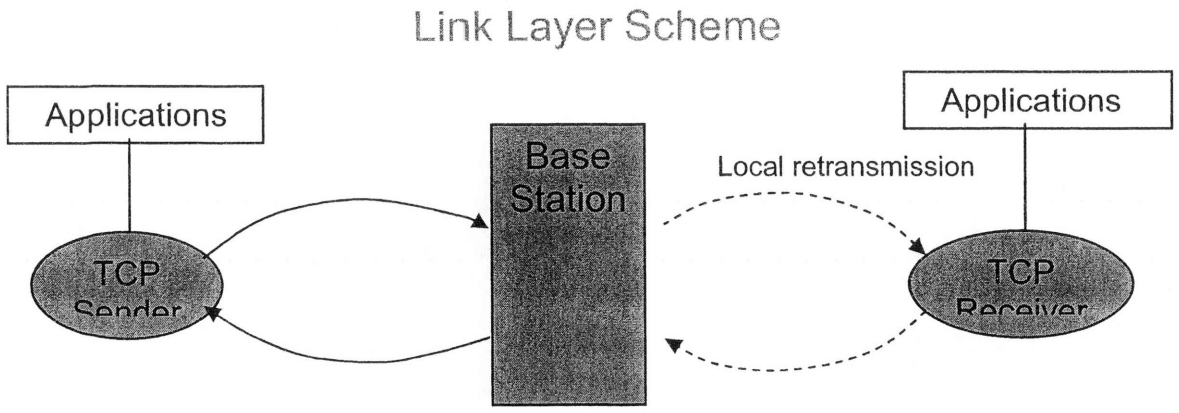


Figure 2.6 Link Layer Scheme

Name	Category	Special Mechanism
E2E	end-to-end	standard TCP-Reno
E2E-NEWRENO	end-to-end	TCP-NewReno
E2E-SACK	end-to-end	selective acks (SACKs)
E2E-ELN	end-to-end	explicit loss notification (ELN)
E2E-ELN-RXMI	end-to-end	ELN with restransmit on first dupack
LL	link-layer	none
LL-TCP-AWARE	link-layer	duplicate ack suppression
LL-SACK	link-layer	SACKs
LL-OPT	link-layer	SACKs and duplicate ack supression
SPLIT	split-connection	none
SPLLT-SACK	split-connection	SACK-based wireless connection

Table 2.1 Summary of protocols

2.6 Summary

This chapter started with descriptions of the best-effort Internet service model and motivated the need for reliable transport protocols discussing the salient features of several protocols of historic interest. We then described the salient features of TCP, the protocol that makes today's Internet reliable and discussed several recently end-to-end proposed end-to-end enhancements to it. In Section 2.4, we surveyed some key concepts of wireless networks. In Section 2.5, we discussed and critiqued conventional approaches to tackling the problem caused by wireless bit-errors and user mobility.

We observe that current approaches towards handling wireless bit-errors, asymmetric effects and small transmission windows do not comprehensively solve observed problems. The material described in this chapter sets the stage for our work. The next chapters will describe in detail the algorithms and simulation results of Wireless Fair Intelligent Congestion Control (WFICC) with and without Explicit Loss Notification (ELN) and Admission Control.

Chapter 3

Introduction about Fair Intelligent Congestion Control and Network Simulation ns-2

3.1 Overview of Fair Intelligent Congestion Control (FICC)

FICC algorithm was originally proposed for ATM networks in [19] and was extended to IP networks later [1]. The main design ideas behind FICC are: First, it tries to maintain the queue length at the bottlenecked router along the path of the session close to a target point to avoid router buffer overflow and underflow (See Figure 3.2). The bottlenecked router always operates at the full capacity of output link without interruption from traffic congestion or buffer starvation. Therefore, the most efficient throughput can be achieved and variations on queue length and consequently queuing delay are reduced. Second, it attempts to allocate the available bandwidth fairly among the sessions competing for the same output link. Each source is continuously informed about its current fair share based on the dynamic network traffic conditions by the feedback control packets. Later the traffic shaper at the source edge

router enforces the individual TCP sender sends the packets at the rate of fair share. The followings describe how FICC algorithm works in the network.

Discovery (RD) packets are generated for each session in proportion to the TCP packets and send to the destination. That is, whenever certain number of TCP packets is generated, there will be one RD packet generated (Figure 3.2). The structure of RD header is illustrated in figure 3.1.

aws	es	dpf	flow_id	direction

Figure 3.1 RD Header structure

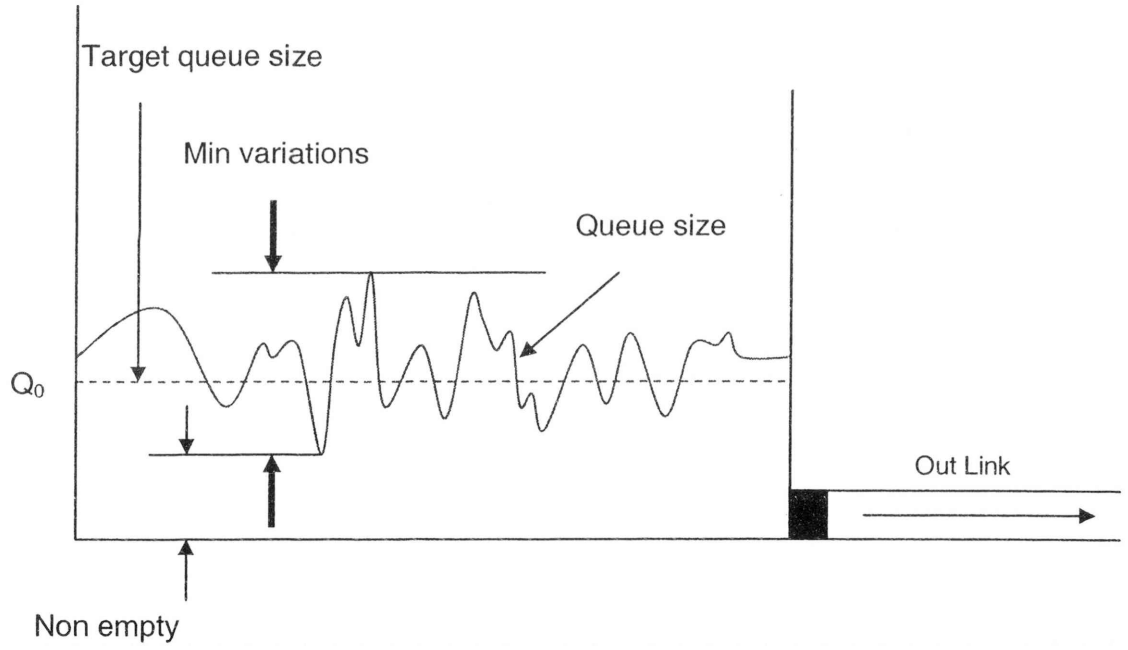


Figure 3.2 Design principle of FICC

At the source edge router as shown in Figure 3.16, special fixed-length Resource Discovery (RD) packets are generated for each session in proportion to the TCP packets and send to the destination. That is, whenever certain number of TCP packets is generated, there will be one RD packet generated. The RD packet is a small, a header-only packet. In its header, it mainly

contains fields of Arrival Packet Rate (APR), Expected Rate (ER) and RD Direction (DIR). The APR field represents the current TCP packet generation rate at the source. The APR field implies the window size that the source TCP will be adapted to based on the network traffic condition. When a RD packet is generated, ER field is assigned current sending rate of traffic shaper at the source edge router.

When receiving RD packet in the forward direction, the router along the path of the session calculates the Mean Arrival Packet Rate (MAPR) and records the queue length at the router. A RD packet will be sent back to the source when it arrives at the destination edge router, namely base station. Having detected RD packet in the backward direction, the router calculates the Explicit Rate (ER) based on the MAPR and buffer fill level in the forward direction and assign ER value to the RD header. When RD arrives at the source edge router, the rate of token bucket at traffic shaper will adapt to the ER value given in the RD header.

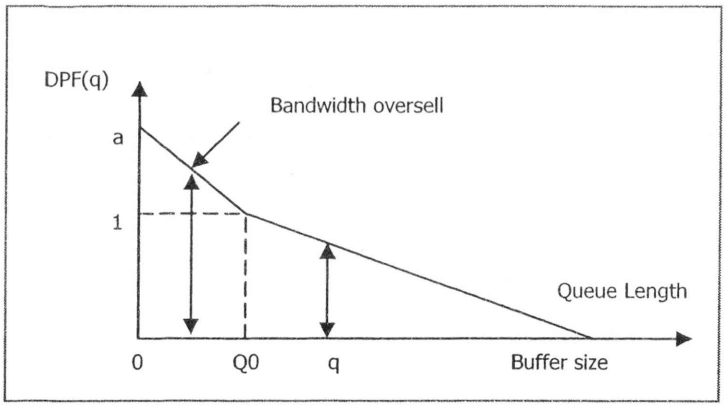


Figure 3.3 Congestion control function of FICC

To calculate the ER at each router along the path of the session, a linear queue control function, called Down Pressure Factor (DPF) is employed in the scheme. The scheme is shown in Figure 3.3 described in details in [11]. The basic characteristics of the function are that it has a value equal to 1 when the queue length is equal to target queue length Q_0, and a value larger/less than 1 when the queue length is less/larger than Q_0. If we define Buffer Utilization Ratio (BUR) as the fixed percentage of the buffer capacity, Q_0 is equal to BUR*

BufferSize. The calculation details are illustrated in Figure 3.4. In Figure 3.4, β (beta) is the average ratio and α (alpha) is the congestion function parameter.

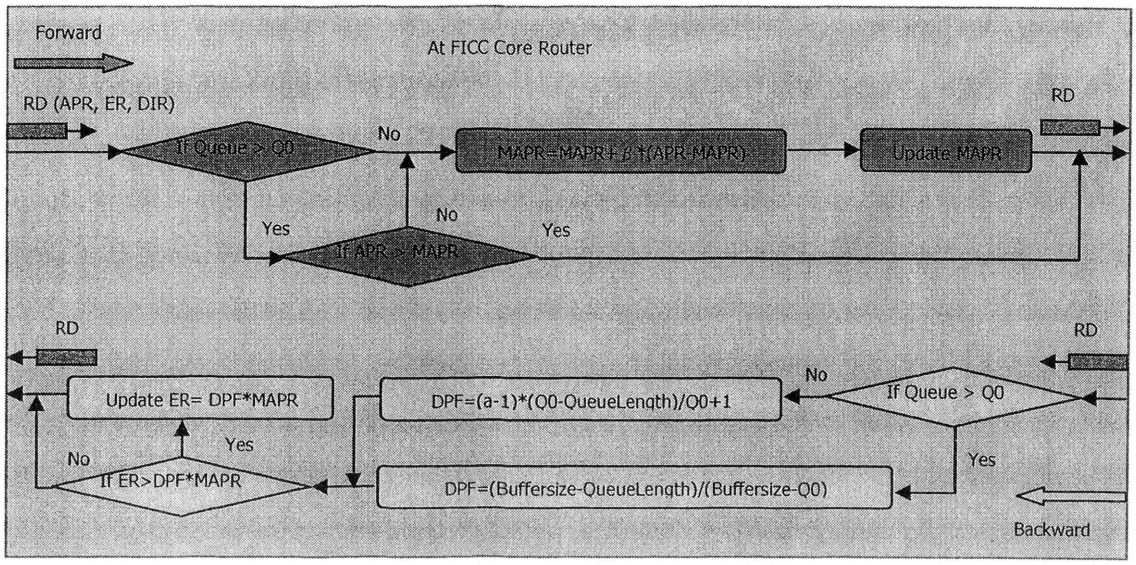


Figure 3.4 Description of FICC Algorithm

3.2 FICC Deployment over Wireless Networks

Transmission Control Protocol (TCP) has been deployed widely in the majority of commercial networks and services. It is thus essential for any wireless networking device that wishes to access these networks and their services to communicate via TCP/IP. Developed some two decades ago, TCP is based on assumptions that hold in wired networks [9], but not necessarily in wireless environment. TCP assumes that packet loss occurs primarily due to congestion somewhere in the network. This assumption obviously does not hold in unreliable high error-rate wireless networks. TCP interprets all packet losses are caused by congestions and it triggers the congestion slow start procedure unnecessarily. As a result, TCP performance is nowhere near as efficient as in wired networks [10].

There are various attempts in solving the problems by researchers [11,12] and can be categorized as three groups: *end-to-end proposals*, *split-connection proposals,* and *link-layer proposals.* . The end-to-end proposals keep a complete TCP connection from source to destination as that in a wired network. This approach includes TCP SACK (Selective

ACKnowledgement) and Explicit Loss Notification (ELN) [13]. TCP SACK can retransmit all known lost packet efficiently. ELN can provide the TCP sender the reasons of packet loss, but there have been no efficient solutions to implement it yet [14]. The split-connection proposals such as Indirect TCP [15] decompose the TCP connection into two sub-connections for the wired and wireless parts of the path. However, this approach does not ensure end-to-end delivery of packets and does not work well if the connection is split several times over the path of the connection. The link-layer proposals combine the ideas of the above two proposals. These protocols remain a complete end-to-end connection but distinguish from the end-to-end proposal by hiding link-loss from TCP sender by using local error recovery. The examples are Snoop [16] and TULIP [17]. Snoop protocol introduces a snoop agent at the base station. The agent monitors every packet that passes in both direction and maintains a cache of TCP segments that have not yet been acknowledged by the receiver. It detects packet loss by duplicate acknowledgements or a local Retransmission TimeOut (RTO) and retransmits the lost packet. However, snoop protocol does not work well in certain wireless environments where a fair RTO is difficult to calculate [18].

Different from the previous research work, this research investigates the relationship between packet loss and network congestion and introduces a feedback based end-to-end congestion control algorithm to the wireless network. In the current Internet architecture, there is no rate feedback from the network to the traffic sources. The control of the network traffic is achieved through the TCP implemented in the end systems. TCP shapes the source traffic by adjusting its transmission window size according to its estimate of the current network status. The strength of this end-system based approach to traffic control is the flexibility and ease of deployment. The drawback is that the source cannot learn the exact state of the network, which leads to less than 'optimum' throughput.

The rate feedback approach considered in this thesis has been tried before in the context of ATM networks (for Available Bit Rate Service or ABR), and it failed. There are two main reasons for it is introduced in this thesis: the edge to edge scheme is acceptable in the Internet that does not require intermediate routers; the approach is scalable and feasible. In previous work [53], the rate feedback can significantly improve the performance of TCP/IP

applications when the available buffering capacity in the network is extremely limited. The source periodically sends probe packets to the destination which then returns these probe packets, carrying the bottleneck fair share in the end to end path, to the source. The [53] simulation results show that rate feedback is quite feasible in TCP/IP. Future work will involve in designing in large scalable networks.

The algorithm is a modification of a Fair Intelligent Congestion Control (FICC) proposed in [19]. The main ideas behind FICC are: First, it tries to maintain the queue length at the bottlenecked router close to a target point to avoid router buffer overflow and underflow. Thus it can avoid congestions and minimize delay variations. Second, it attempts to allocate the available bandwidth fairly among the sessions competing for the same output link. The innovation of the algorithm presented in this paper is to modify the original FICC in a way such that the queue lengths can be effectively controlled when it is jointly employed with TCP in the wireless network. It will be shown in this paper that our Window-based Fair Intelligent Congestion Control (WFICC) scheme can avoid congestions, reduce delay variations, achieve fairness and improve the effective throughput in the wired-cum-wireless networks.

3.3 Network Simulation NS-2

3.3.1 Overview

Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in

collaboration with other researchers including ACIRI. Ns has always included substantial contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

NS is an event driven network simulator developed at UC Berkeley that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. The NS project is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats.

As shown in Figure 3.5, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object). In other words, to use NS, we program in OTcl script language. To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler.

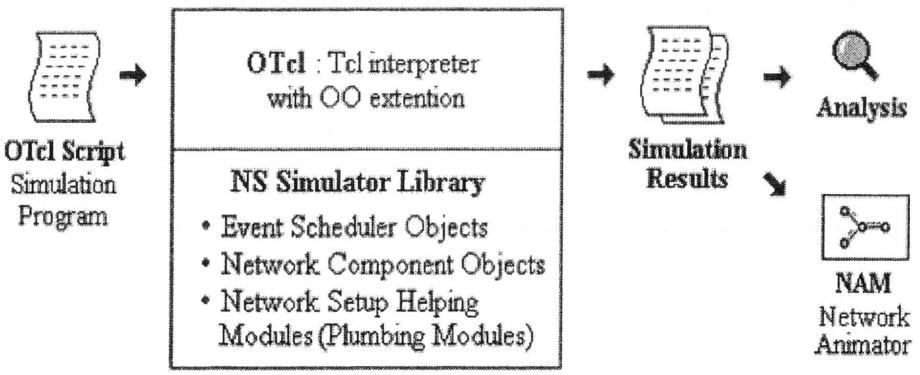


Figure 3.5 Simplified User's View of NS

Another major component of NS beside network objects is the *event scheduler*. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event. Network components communicate with one another passing packet, however this does not consume actual simulation time. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. Another use of an event scheduler is timer. For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does. The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

As mentioned earlier, NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object. The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object (not in the data path) can be entirely implemented in OTcl. Figure 3.1.2 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an

OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

Figure 3.7 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is an OO extended Tcl interpreter with network simulator libraries.

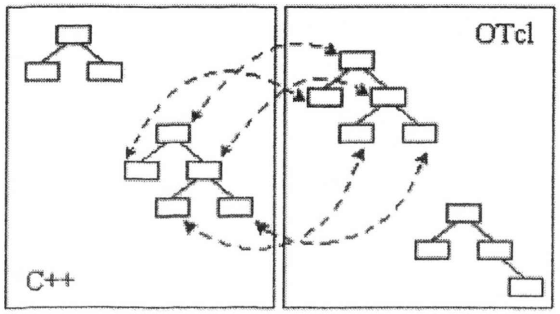


Figure 3.6 C++ and OTcl: The Duality

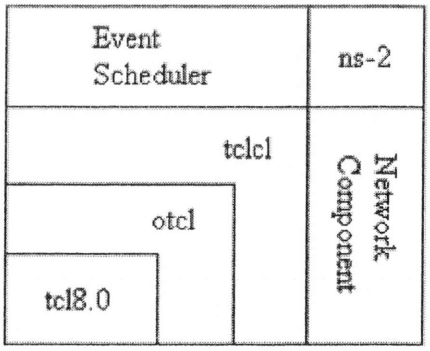


Figure 3.7 Architectural View of NS

At this point, we might be wondering about how to obtain NS simulation results. As shown in Figure 3.1, when a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl (or more

specifically, OTcl) script. The data can be used for simulation analysis (two simulation result analysis examples are presented in later sections) or as an input to a graphical simulation display tool called *Network Animator (NAM)* that is developed as a part of VINT project. NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

3.3.2 Network Components

Figure 3.8 shows a partial OTcl class hierarchy of NS, which will help understanding the basic network components. For a complete NS class hierarchy, visit *http://www-sop.inria.fr/rodeo/personnel/Antoine.Clerget/ns*.

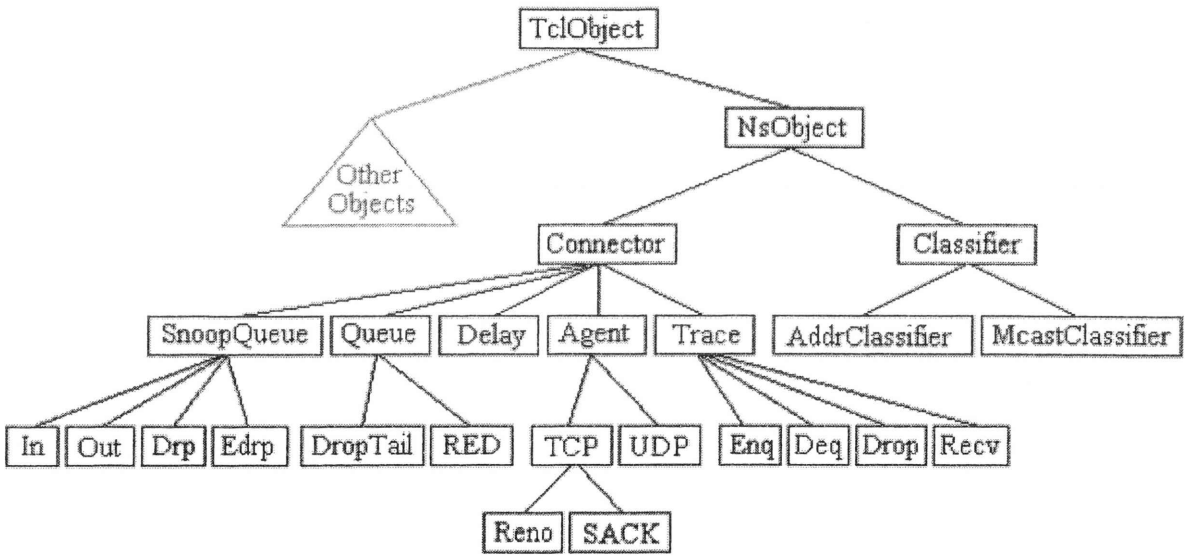


Figure 3.8 Class Hierarchies (Partial)

The root of the hierarchy is the TclObject class that is the superclass of all OTcl library objects (scheduler, network components, timers and the other objects including NAM related ones). As an ancestor class of TclObject, NsObject class is the superclass of all basic network component objects that handle packets, which may compose compound network objects such as nodes and links. The basic network components are further divided into two subclasses,

Connector and Classifier, based on the number of the possible output data paths. The basic network objects that have only one output data path are under the Connector class, and switching objects that have possible multiple output data paths are under the Classifier class.

- **Node and Routing**

A node is a compound object composed of a node entry object and classifiers as shown in Figure 3.9. There are two types of nodes in NS. A unicast node has an address classifier that does unicast routing and a port classifier. A multicast node, in addition, has a classifier that classify multicast packets from unicast packets and a multicast classifier that performs multicast routing.

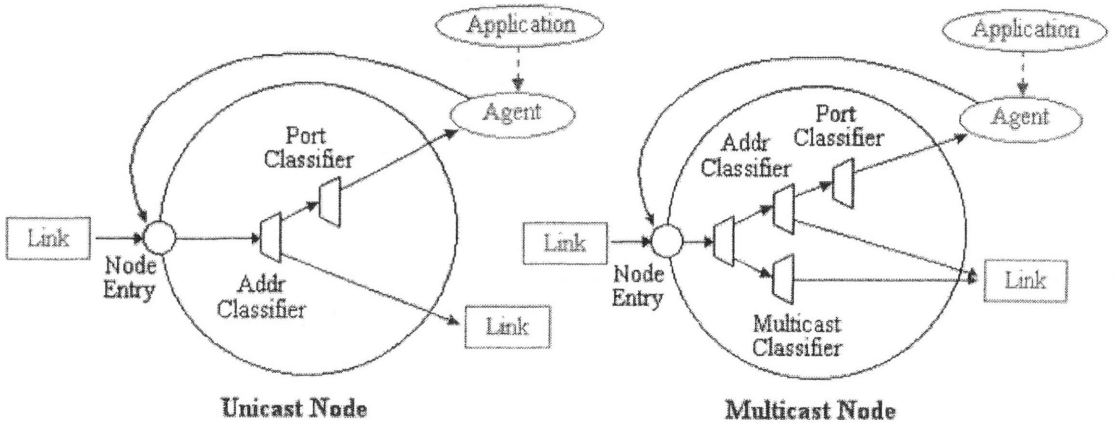


Figure 3.9 Node (Unicast and Multicast)

In NS, Unicast nodes are the default nodes. To create Multicast nodes the user must explicitly notify in the input OTcl script, right after creating a scheduler object, that all the nodes that will be created are multicast nodes. After specifying the node type, the user can also select a specific routing protocol other than using a default one.

✓ ***Unicast***

- *$ns* rtproto *type*
- *type*: Static, Session, DB, cost, multi-path

✓ ***Multicast***

- *$ns* multicast (right after set *$ns* [new Scheduler])
- *$ns* mrtproto *type*
- *type*: CtrMcast, DM, ST, BST

- **Link**

A link is another major compound object in NS. When a user creates a link using a duplex-link member function of a Simulator object, two simplex links in both directions are created as shown in Figure 3.10.

One thing to note is that an output queue of a node is actually implemented as a part of simplex link object. Packets dequeued from a queue are passed to the Delay object that simulates the link delay, and packets dropped at a queue are sent to a Null Agent and are freed there. Finally, the TTL object calculates Time To Live parameters for each packet received and updates the TTL field of the packet.

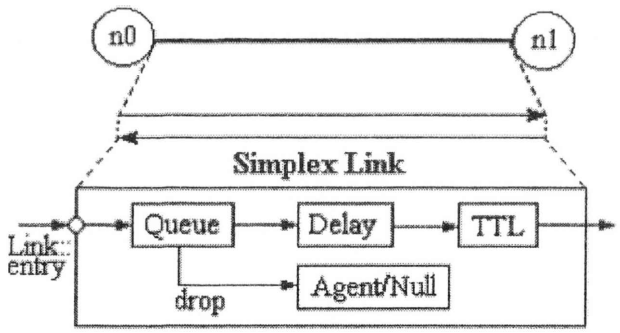


Figure 3.10 Link

- **Tracing**

In NS, network activities are traced around simplex links. If the simulator is directed to trace network activities (specified using *$ns* trace-all *file* or *$ns* namtrace-all *file*), the links created after the command will have the following trace objects inserted as shown in Figure 3.11. Users can also specifically create a trace object of type *type*

between the given *src* and *dst* nodes using the create-trace {*type file src dst*} command.

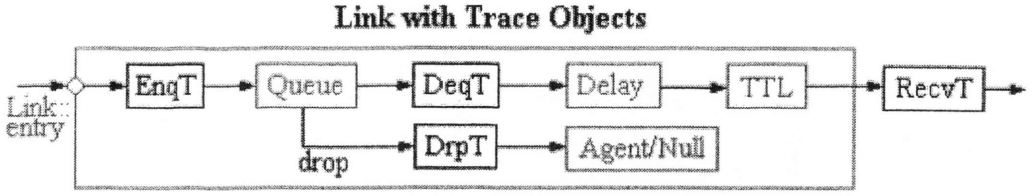


Figure 3.11 Inserting Trace Objects

When each inserted trace object (i.e. EnqT, DeqT, DrpT and RecvT) receives a packet, it writes to the specified trace file without consuming any simulation time, and passes the packet to the next network object. The trace format will be examined in the General Analysis Example section.

- **Queue Monitor**

Basically, tracing objects are designed to record packet arrival time at which they are located. Although a user gets enough information from the trace, he or she might be interested in what is going on inside a specific output queue. For example, a user interested in RED queue behavior may want to measure the dynamics of average queue size and current queue size of a specific RED queue (i.e. need for queue monitoring). Queue monitoring can be achieved using queue monitor objects and snoop queue objects as shown in Figure 3.12.

When a packet arrives, a snoop queue object notifies the queue monitor object of this event. The queue monitor using this information monitors the queue. A RED queue monitoring example is shown in the RED Queue Monitor Example section. Note that snoop queue objects can be used in parallel with tracing objects even though it is not shown in the below figure.

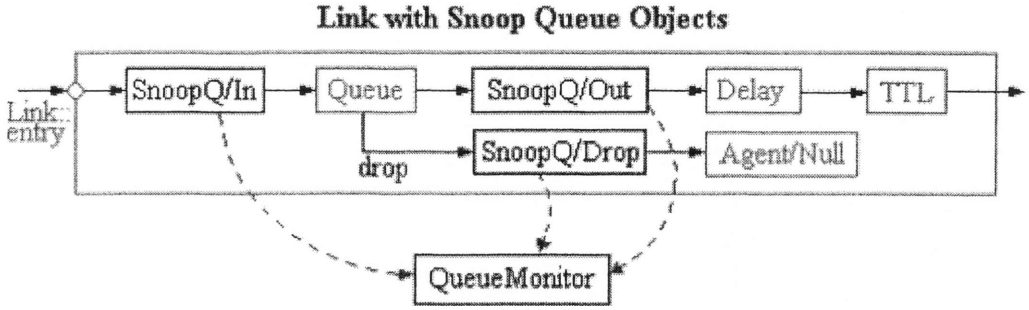


Figure 3.12 Monitoring Queue

- **Packet**

A NS packet is composed of a stack of headers, and an optional data space (see Figure 3.10). As briefly mentioned in the "Simple Simulation Example" section, a packet header format is initialized when a Simulator object is created, where a stack of all registered (or possibly useable) headers, such as the common header that is commonly used by any objects as needed, IP header, TCP header, RTP header (UDP uses RTP header) and trace header, is defined, and the offset of each header in the stack is recorded. What this means is that whether or not a specific header is used, a stack composed of all registered headers is created when a packet is allocated by an agent, and a network object can access any header in the stack of a packet it processes using the corresponding offset value.

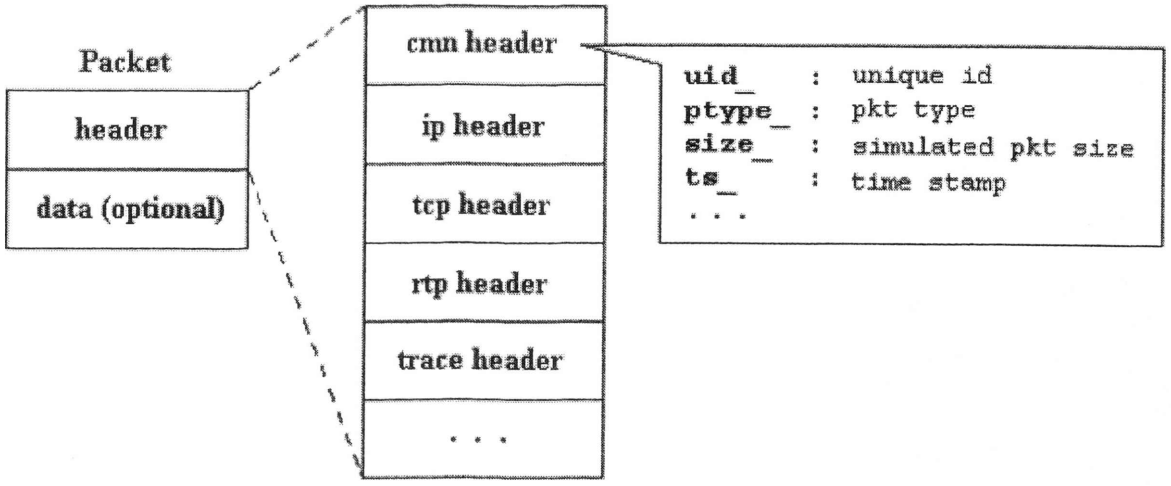


Figure 3.14 NS Packet Format

Usually, a packet only has the header stack (and a data space pointer that is null). Although a packet can carry actual data (from an application) by allocating a data space, very few application and agent implementations support this.

3.3.3 Structure of Network Simulation Version 2 (NS-2)

Figure 3.15 shows a part of the directory structure of the simulator if we installed it using the ns-allinone-2.1b package.

Among the sub-directories of ns-allinone-2.1b, ns-2 is the place that has all of the simulator implementations (either in C++ or in OTcl), validation test OTcl scripts and example OTcl scripts. Within this directory, all OTcl codes and test/example scripts are located under a sub-directory called tcl, and most of C++ code, which implements event scheduler and basic network component object classes, except the WWW related ones, are located in the main level.

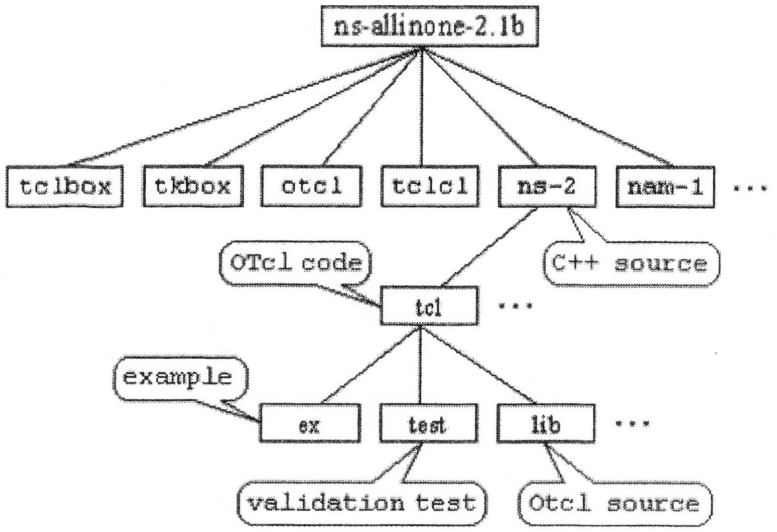


Figure 3.15 NS Directory Structure

The tcl directory has sub-directories, among which the lib directory that contains OTcl source codes for the most basic and essential parts of the NS implementation (agent, node, link, packet, address, routing, and etc.) is the place one as a user or as a developer will visit the most. Note that the OTcl source codes for LAN, Web, and Multicast implementations are located in separate subdirectories of tcl. Following is a partial list of files in "ns-2/tcl/lib" directory.

- ns-lib.tcl: The simulator class and most of its member function definitions except for LAN, Web, and Multicast related ones are located here.
- ns-default.tcl: The default values for configurable parameters for various network components are located here. Since most of network components are implemented in C++, the configurable parameters are actually C++ variables made available to OTcl via an OTcl linkage function, bind(*C++_variable_name*, *OTcl_variable_name*).
- ns-packet.tcl: The packet header format initialization implementation is located here. When we create a new packet header, we should register the header in this file to make the packet header initialization process to include our header into the header stack format and give us the offset of our header in the stack.
- other OTcl files: Other OTcl files in this directory, contain OTcl implementation of compound network objects or the front end (control part) of network objects in C++.

The FTP application is entirely implemented in OTcl and the source code is located in "ns-source.tcl".

Two sub-directories of tcl that might be interesting for a user who wants to know how to design a specific simulation are ex and test. The former directory contains various example simulation scripts and the latter contains simulation scripts that validate the NS installed in our machine by running various simulations and comparing the results with the expected results.

3.4 General Simulation Setup

The simulation used is Network Simulation ns2 [2] to evaluate FICC protocol in wireless networks. Throughput, goodput, congestion window and queue length are metrics for comparing the networks without and with FICC. The results were obtained in both wired and wireless networks.

Calculation of goodput and throughput (based on [1])

$$\text{Goodput} = \frac{\text{Number of bytes sent} - \text{Number of data bytes retransmitted}}{\text{Time}}$$

$$\text{Throughput} = \frac{\text{Number of bytes sent}}{\text{Time}}$$

where Time = Time + period (period is 0.01 secs)

Time is obtained by taking the current time

Number of bytes sent = data bytes + RD bytes

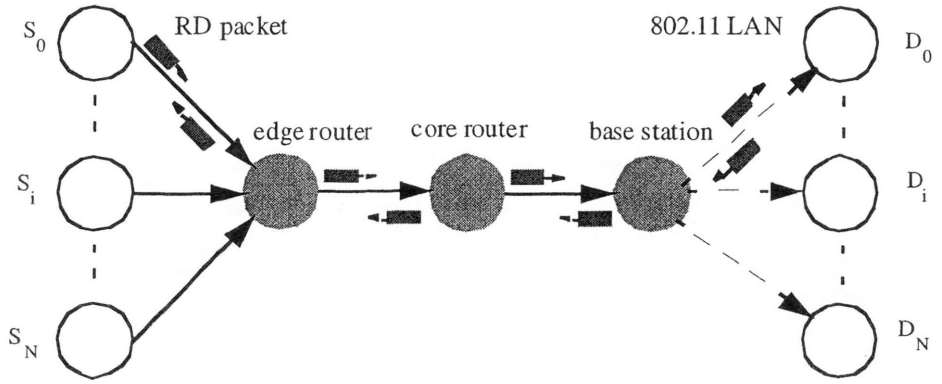


Figure 3.16 Illustration of a simple hybrid network and WFICC control loop

- The scenario is set up as above
- S0-S4: sources
- E1, E2: edge routers
- C1: core router
- D1-D4: destinations

- TCP0-TCP4: TCP agents attached to S0-S4, respectively
- TCKSink0-TCPSink4: TCKsink agents attached to D1-D4, respectively
- S0-S4 will be connected to D1-D4, respectively
- ficcVAR1-VAR3: agents attached to E1, C1, E2 respectively to store information
- An RD agent is attached to each source and destination to send and receive RD packets

The FICC algorithm works as follows:

- Every time a TCP packet from S0-S9 passes each router, it saves the queue length in ficcVAR agent.

- An RD byte (DPF, ES, AWS, direction) will be generated for every RD_RATE TCP bytes for each flow.
- As RD packet passes each router toward destination, MAWS in ficcVAR will be update by

If (QueueLength > Q_0)

If (AWS < MAWS)

$$MAWS = MAWS + \beta * (AWS - MAWS)$$

else

$$MAWS = MAWS + \beta * (AWS - MAWS)$$

- When RD packet arrives destination, it will be returned to source with direction field of FORWARD replaced by BACKWARD.
- When backward RD packet passes edge router E2 on the way back to source, queue length is extracted from ficcVAR 3 (attached to E2) to calculate new DPF (the old DPF currently stored in ficcVAR is initialised to 0) by

If (QueueLength > Q_0)

$$DPF = \frac{BufferSize - QueueLength}{BufferSize - Q_0}$$

else

$$DPF = \frac{(alpha - 1) * (Q_0 - QueueLength)}{Q_0} + 1$$

where alpha = 1.15

Buffer Size = mean_packetSize_ * qlim_ = 500 * 50 = 25000 bytes

Q_0 : Target Point = Buffer Size * BUR = 25000 * 0.3 = 7500 bytes

- Calculate new ES:
    - ES = DPF (new) * MAWS
- At E2, new ES and new DPF will be updated in both ficcVAR3 and RD header fields.
- At C1 and E1, the same calculations are carried and choose min(oldDPF, new DPF) and min(oldES, newES) to update both ficcVAR and RD header.
- When an RD packet arrives the source, ES will be extracted to compute win_ficc (new advertised window); then DPF is used to determine new usable congestion window (window_ficc_).
- Each ACK also carried ES and DPF in its header, every time it passes E1, those fields will be filled with the most updated information, therefore advertised (calculated based on those information) will be update more frequently.

3.5. Summary

This chapter gave the overview of the Fair Intelligent Congestion Control (FICC) protocol that was previously developed by the ARN (Advanced Research Networking) Group, UTS. FICC algorithm was successfully working on ATM and was extended to IP networks later. The idea of how to apply FICC in the wireless networks was generally mentioned.

Network Simulation version 2 (ns-2) was used to implement the algorithm and was also comprehensively described. The general setup for all the implementations was shown in detail.

Chapter 4

Fair Intelligent Congestion Control in Wired-Cum-Wireless Networks (WFICC)

4.1 Introduction

As mentioned in chapter 1, TCP assumes that packet losses are mainly due to congestion somewhere in the network. Once a loss condition is detected, it resets its sending window and triggers the slow-start procedure. This assumption does not hold in unreliable high error-rate wireless networks. As a result, TCP performance is nowhere near as efficient as in wired

networks. This chapter isolates packet loss due to congestion and introduces a Window-based Fair Intelligent Congestion Control (WFICC) scheme to improve TCP performance. Simulations results demonstrate that by eliminating buffer overflows caused by congestions, WFICC provides better goodput, improves the delay and fairness of TCP connections significantly.

This algorithm is a modification of a Fair Intelligent Congestion Control (FICC) proposed in [1]. The main ideas behind FICC are: First, it tries to maintain the queue length at the bottlenecked router close to a target point to avoid router buffer overflow and underflow. Thus it can avoid congestions and minimize delay variations. Second, it attempts to allocate the available bandwidth fairly among the sessions competing for the same output link. The innovation of the algorithm presented in this paper is to modify the original FICC in a way such that the queue lengths can be effectively controlled when it is jointly employed with TCP in the wireless network. It will be shown in this paper that our Window-based Fair Intelligent Congestion Control (WFICC) scheme can avoid congestions, reduce delay variations, achieve fairness and improve the effective throughput in the wired-cum-wireless networks.

4.2 Operation

4.2.1 At sources

FICC operates by employing a special Resource Discovery (RD) packet which is generated at each source for each flow proportionally based on the packet arrive rate.

RD packets generated at sources for each flow proportionally based on window arrival rate. The RD packet will carry the estimated arrival window rate (aws) in its APR field. The destinations will receive and return RD-packet back to sources. The explicit rate es value in RD packet will then be used to adjust TCP congestion window.

4.2.2 At routers

FICC is employed at each router along the network path to estimate a fair share of bandwidth for competing TCP connections, calculate available network capacity and feedback relevant information to the sources. Each router along the path directly updates feedback information concerning the network conditions in the RD packets when they pass in the forward and backward direction. In order to estimate the current traffic generation rate of the network and allocate it among connections fairly, a Mean Allowed Window Rate (MAWS) is kept at each router.

$$MAWS = MAWS + \beta * (AWS - MAWS)$$

where AWS is the value of current window rate arrived and AWS field of the arriving forward RD packet. MAWS represents an estimate of the average load passing through the router at the current time. When the network operates at the actable level, the correspondent MAWS is considered as the optimal window rate for each flow.

The destinations will send back RD packets to the sources. Upon receiving backward RD packet, it passes the feedback algorithm that is responsible for matching the TCP sending rate to the expected rate supported by the network.

4.3 Simulation Results and Analysis

The simulation used is Network Simulation ns2 [2] to evaluate FICC protocol in wireless networks. Throughput, goodput, congestion window and queue length are metrics for comparing the networks without and with FICC. The results were obtained in both wired and wireless networks.

4.3.1 WFICC on Wired Networks

The wired network is set up as in Figure 4.1 below

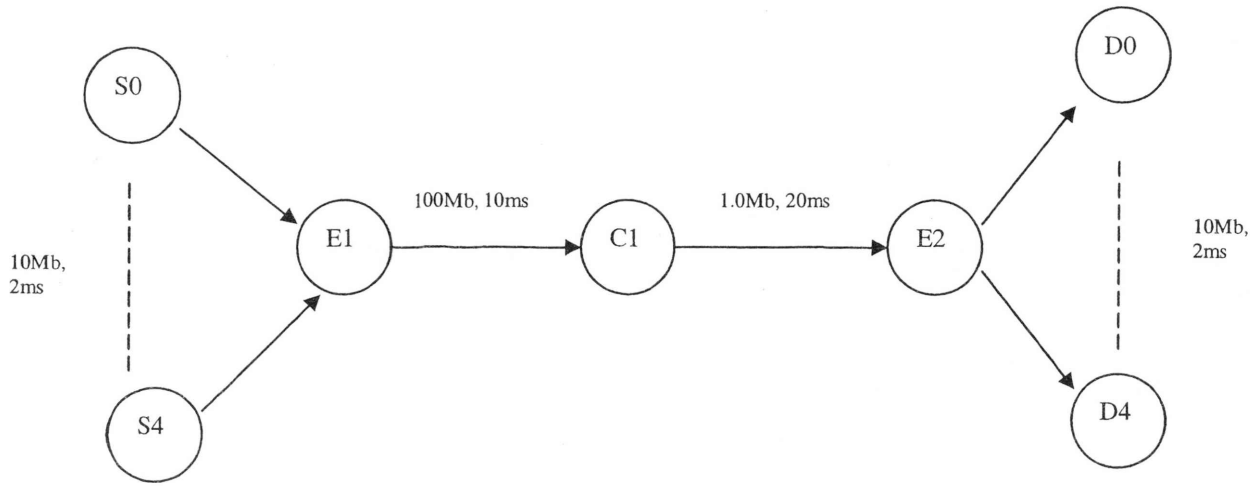


Figure 4.1 FICC on wired network scenario

The FICC algorithm works as follows:

- Every time a TCP packet from S0-S9 passes each router, it saves the queue length in ficcVAR agent.
- An RD byte (DPF, ES, AWS, direction) will be generated for every RD_RATE TCP bytes for each flow.
- As RD packet passes each router toward destination, MAWS in ficcVAR will be update by

```
If (QueueLength > Q0)
        If (AWS < MAWS)
                MAWS = MAWS + β * (AWS – MAWS)
else
        MAWS = MAWS + β * (AWS – MAWS)
```

- When RD packet arrives destination, it will be returned to source with direction field of FORWARD replaced by BACKWARD.

- When backward RD packet passes edge router E2 on the way back to source, queue length is extracted from ficcVAR 3 (attached to E2) to calculate new DPF (the old DPF currently stored in ficcVAR is initialised to 0) by

  If (QueueLength > Q_0)

$$DPF = \frac{BufferSize - QueueLength}{BufferSize - Q_0}$$

  else

$$DPF = \frac{(alpha - 1) * (Q_0 - QueueLength)}{Q_0} + 1$$

  where alpha = 1.15

  Buffer Size = mean_packetSize_ * qlim_ = 500 * 50 = 25000 bytes

  Q_0 : Target Point = Buffer Size * BUR = 25000 * 0.3 = 7500 bytes

- Calculate new ES:
  - ES = DPF (new) * MAWS

- At E2, new ES and new DPF will be updated in both ficcVAR3 and RD header fields.
- At C1 and E1, the same calculations are carried and choose min(oldDPF, new DPF) and min(oldES, newES) to update both ficcVAR and RD header.
- When RD arrives source, ES will be extracted to compute win_ficc (new advertised window); then DPF is used to determine new usable congestion window (window_ficc_).

- Each ACK also carried ES and DPF in its header, every time it passes E1, those fields will be filled with the most updated information, therefore advertised (calculated based on those information) will be update more frequently.

Parameters Summary

- alpha = 1.2
- beta = 0.1
- gamma = 2
- BUR = 0.2
- RD packet size = 50 bytes
- TCP packet size = 500 bytes
- RD rate = 50 → every 50 bytes of date packet generated, 1 byte of RD will be generated, so in order to have one RD packet generated, there should be 50x50 bytes = 2500bytes of data generated. Since TCP packet size is 500 bytes, one RD packet will be generated for every 5 TCP packets
- TCP type: TCP-Reno
- Simulation time = 200 seconds
- Starting recording time: at 50s
- Number of sources: 5
- Number of destinations: 5
- Bottleneck bandwidth (E2 to each destination): 1Mpbs
- Queue Limit = 50pkt = 50 * 500bytes = 25Kbytes
- Target point = Queue Limit x BUR = 25Kbytes x 0.e = 5Kbytes

Simulation Results

- Queue Length

The main reason for TCP degradation is due to the overflow of buffer at bottleneck router. TCP doesn't assume any explicit signaling of congestion state from the underlying network. It only considers the congestion state of the network implicitly such as the arrival of acknowledgements (ACKs), timeouts, and the receipt of duplicate ACKs. Therefore, the window-based congestion control mechanisms of TCP may interact with the underlying network in undesirable ways, resulting severe congestion, degraded throughput and unfairness.

Figure 4.2 shows queue length at bottleneck with DropTail, it varies tremendously between 0 and maximum buffer size. In Figure 4.2, we can see the queue length vibrates around operating target point (5Kb). This achieves the main idea of FICC to keep the queue length around operating target point.

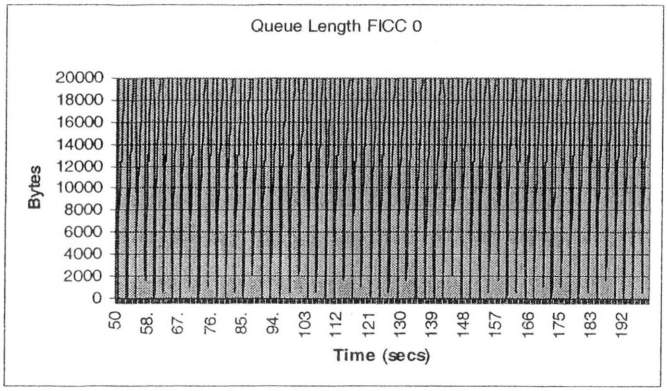


Figure 4.2 Queue length at bottleneck without FICC (DropTail)

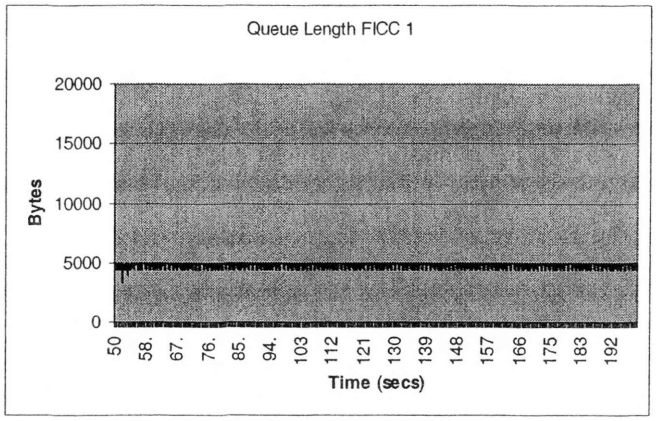


Figure 4.3 Queue length at bottleneck with FICC queue

- Congestion Window

Figure 4.4 shows the congestion window with TCP only, illustrating that window keeps dropping due to congestion and timeout very frequently, resulting degraded goodput. When FICC applies in Figure 4.5, the congestion window is varying around the size of 6.5 pkt size, not dropping to zero, so the goodput is increasing substantially

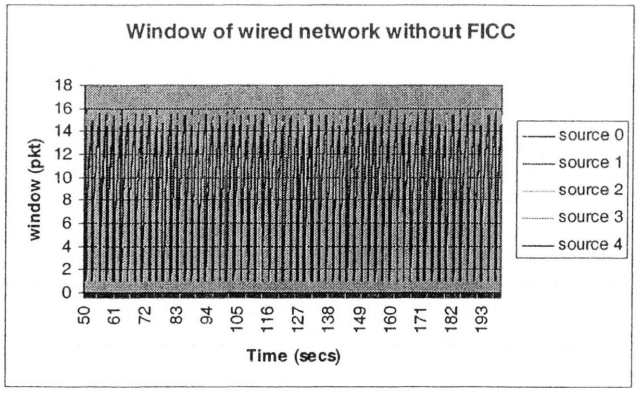


Figure 4.4 Congestion window size without FICC

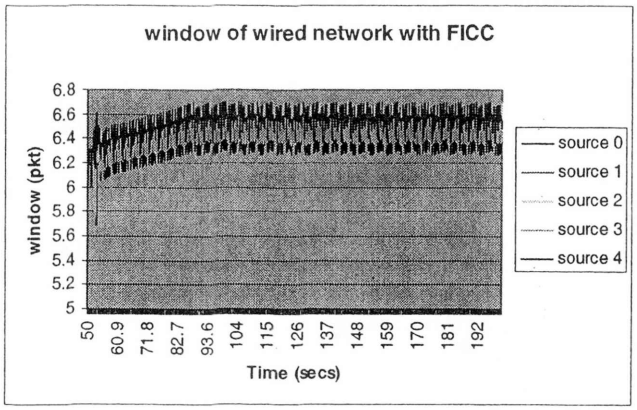


Figure 4.5 Congestion window size with FICC

As the queue length is quite stable, the resulting congestion window size is also stable

- Goodput

Since the bandwidth at bottleneck is 1Mbps, each source shares maximum bandwidth of 200Kbps. In case of without FICC, all sources converge at around 180Kbps and 185Kbps. In other words, the bandwidth of 1Mbps is not fairly shared among TCP flows due to congestion while in the case of FICC's present, all TCP flows are closely matched at 185Kbps.

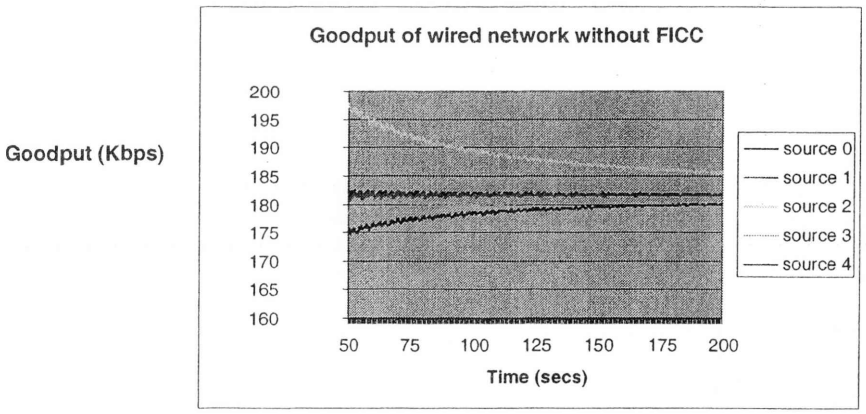


Figure 4.6 Goodput without FICC

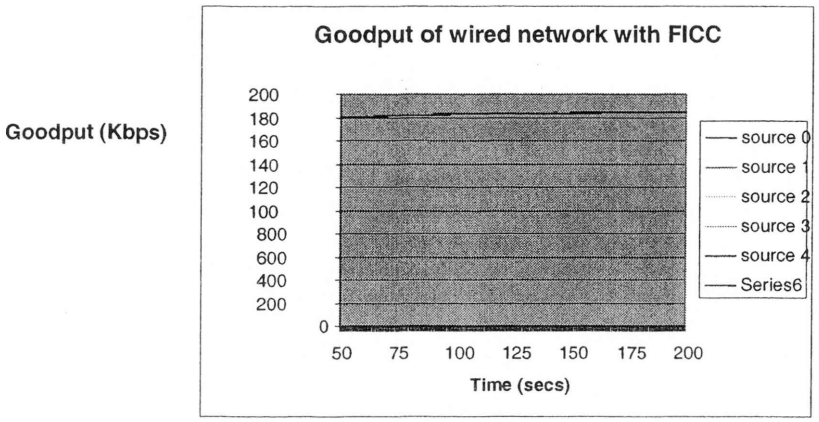


Figure 4.7 Goodput with FICC

4.3.2 WFICC on Wireless Networks

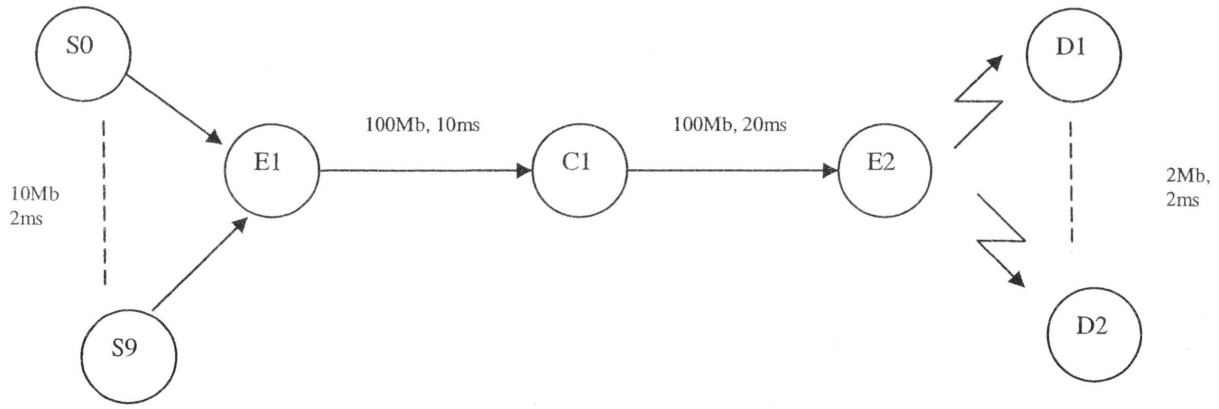


Figure 4.8 FICC on wired-cum-wireless network scenario

The scenario for wireless is set up as wired part except there are no links between E2 and destinations; instead it runs on top of an IEEE802.11 MAC sub-layer.

The setup is similar to wired network, but we have to consider

1. Since this is a combined wired and wireless networks, we need to use hierarchical routing in order to route packets between wireless and wired domains. The routing information for wired nodes on connectivity of the topology, i.e., how are nodes connected to one another through Links. This connectivity information is used to populate the forwarding tables in each wired nodes. However wireless nodes have no "links". Packets are routed in a wireless topology using their adhoc routing protocols which build forwarding tables by exchanging routing queries among its neighbours. So in order to exchange packets among these wired and wireless nodes, we used E2 as base-station acting as gateways between the two domains. We put wired and wireless nodes in different domains. Domains and clusters are defined by means of hierarchical topology structure as:

```
# set up for hierarchical routing
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2          ;# number of domains
lappend cluster_num 1 1      ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 12 11       ;# number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain
```

2. In order to create base-station E2, we need to configure the node. Since E2 is a gateway between wired and wireless domains, they need to have wired routing mechanism turned on by setting node-config option –wiredRouting ON, it will be OFF later for configuring mobile nodes.
3. The base-station E2 has to be in the same domain as wireless nodes because all packets originating from the wired domain, and destined for a wireless nodes will reach base-station E2 which then uses its adhoc routing protocol to route the packet to its correct destination
4. The rest of the setup is similar to wired part

Parameters Summary

- alpha = 2
- beta = 0.9
- gamma = 2
- BUR = 0.2
- RD packet size = 50 bytes
- TCP packet size = 500 bytes
- RD rate = 300 → every 300 bytes of date packet generated, 1 byte of RD will be generated, so in order to have one RD packet generated, there should be 50x300 bytes = 15000bytes of data generated. Since TCP packet size is 500 bytes, one RD packet will be generated for every 30 TCP packets
- TCP type: TCP-Reno
- Simulation time = 200 seconds
- Starting recording time: at 50s
- Number of sources: 5
- Number of destinations: 5
- Bottleneck bandwidth = EEE 802.11 wireless links bandwidth = 2 Mbps
- Queue Limit = 50Pkt = 50x500bytes = 25KB
- Target point = Queue Limit x BUR = 25Kbytes * 0.2 = 5Kbytes

4.3.2.1 No Errors (PER = 0)

- Queue Length

This is similar to the case of Figure 4.2 and 4.3, the queue length variation is a bit larger in its counterpart of wired networks due to dropping packet in wireless links

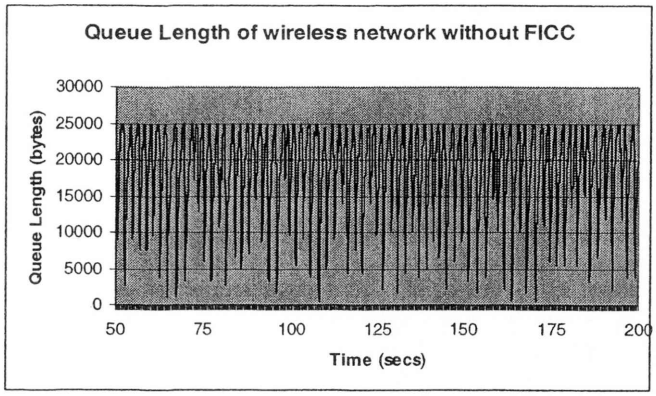


Figure 4.9 Queue length without FICC

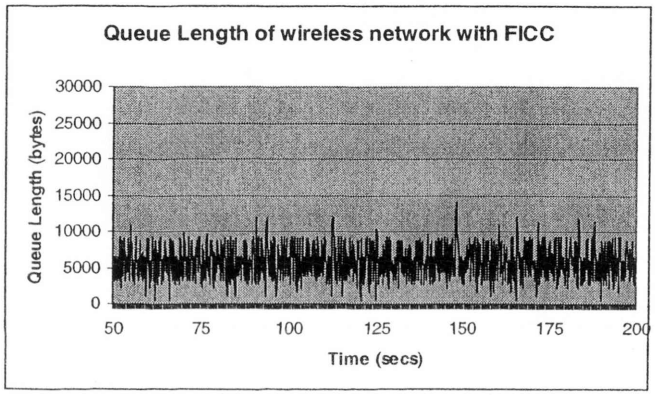


Figure 4.10 Queue length with FICC

- Congestion Windows

Figure 4.11 and 4.12 are similar to the case of wired networks (Figure 4.4 and 4.5). FICC controls the traffic well on wireless links so that no packet drops resulting no window reduction and achieving better goodput.

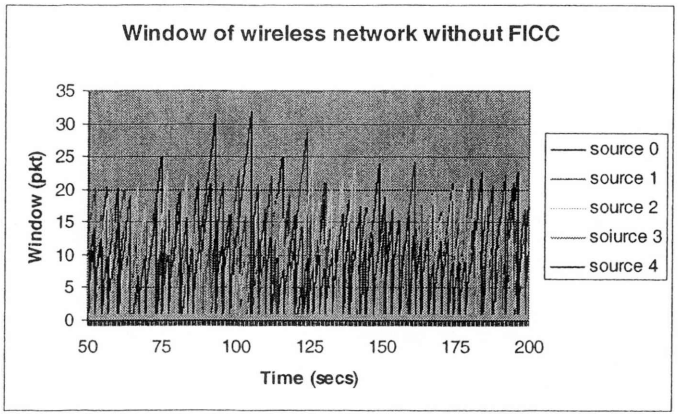


Figure 4.11 Congestion window without FICC

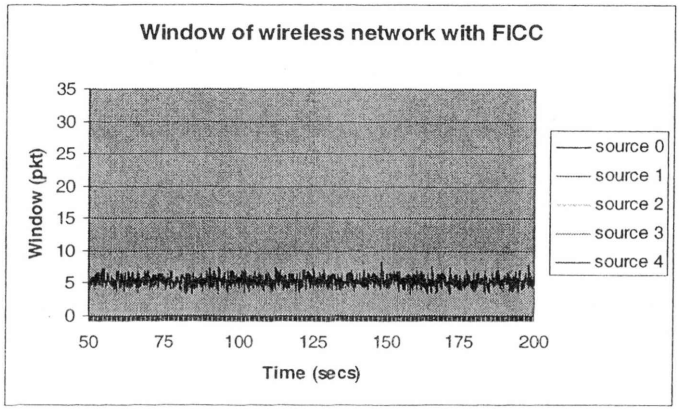


Figure 4.12 Congestion window with FICC

- Goodput

The bandwidth in EEE802.11 used for simulation is 2Mbps but the actually bandwidth for raw data is around 1.3Mbps. As the figures suggest, each TCP flows occupies around 150Kps and total bandwidth of five sources is around 750Kbp which is quite low compared the expected bandwidth of 1.3Mpb. We are working on this to improve the throughput for each flow. So far with the present of FICC, all TCP flows share the bandwidth evenly though still lower than the expectation.

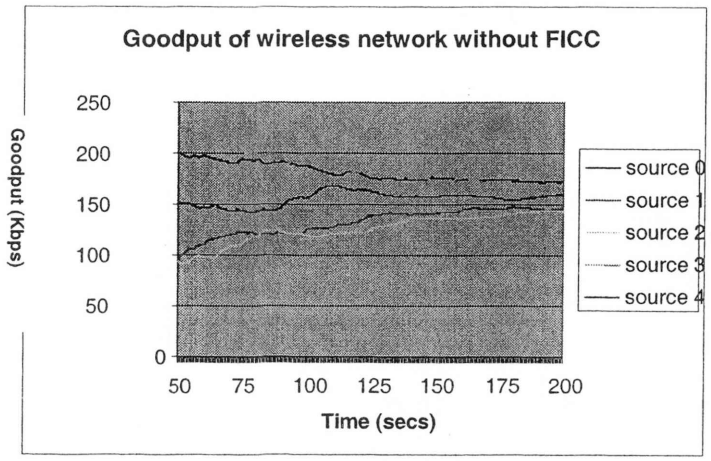


Figure 4.13 Goodput without FICC

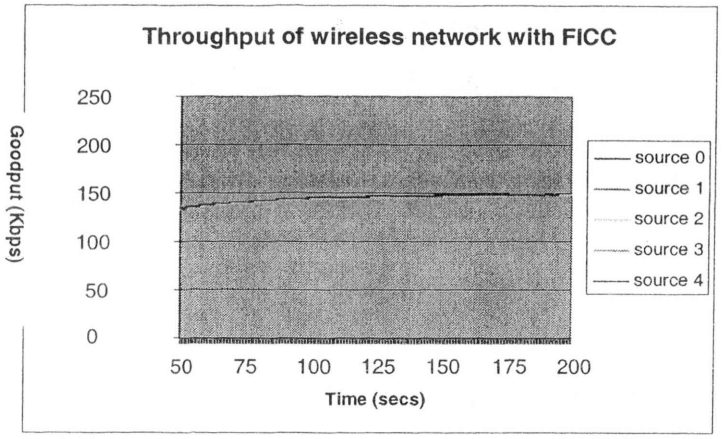


Figure 4.14 Goodput with FICC

4.3.2.2 With Errors

Wireless networks suffer BER due to interference … The parameters are the same in the previous case, but now the wireless link (base station and mobile destination nodes) produce errors with different BERs

Base station (E2): error rate = 0.0005

Mobile destination d0: rate = 0.001

Mobile destination d1: rate = 0.002

Mobile destination d2: rate = 0.003

Mobile destination d3: rate = 0.004

Mobile destination d4: rate = 0.005

- Queue Length

When errors occur, the queue length variation is larger than the case of no error due to more packet drops in wireless links, but the queue length is still not overflow or empty as in the case without FICC

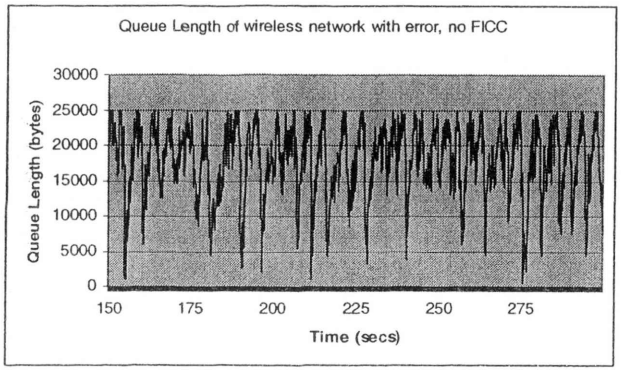


Figure 4.15 Queue Length without FICC

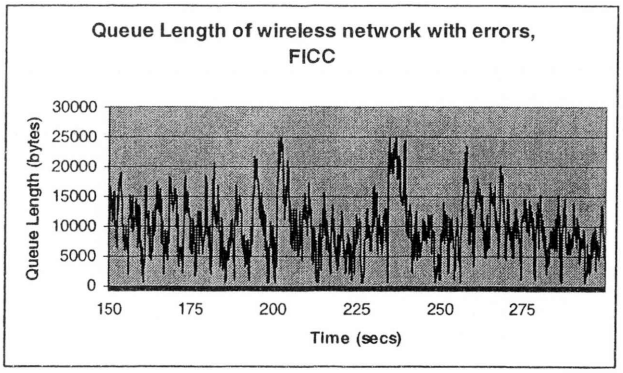


Figure 4.16 Queue Length with FICC

- Congestion Window

Obviously, in the present of errors, more packet drops occur in the wireless links, the congestion window drops more often due to misinterpretation of being congestion. When FICC applies, the traffic is well control, and no packet drops due to congestion, therefore window reduction due to congestion is eliminated as illustrated in figures 4.17 and 4.18; better goodput and fairness are achieved as illustrated in Figures 4.19 and 4.20

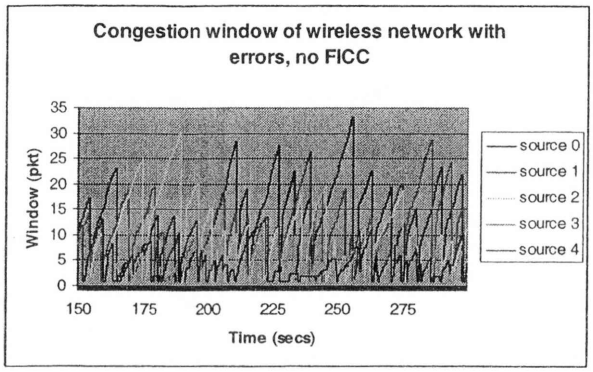


Figure 4.17 Window without FICC

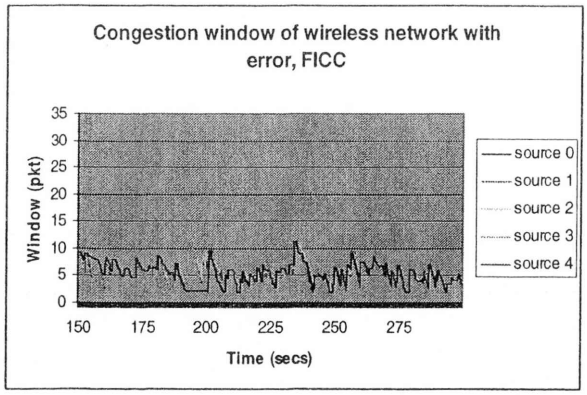


Figure 4.18 Window with FICC

- Goodput

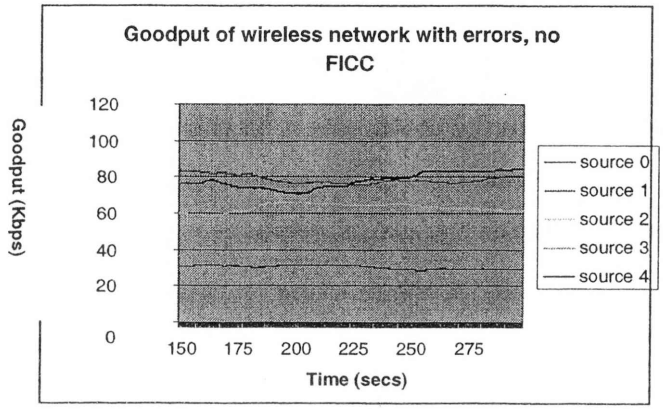


Figure 4.19 Goodput without FICC

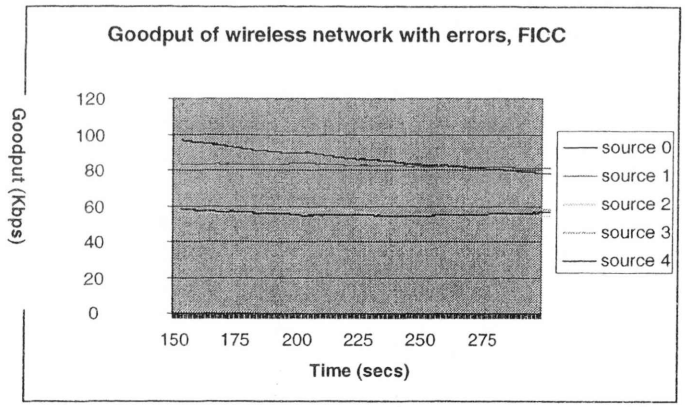
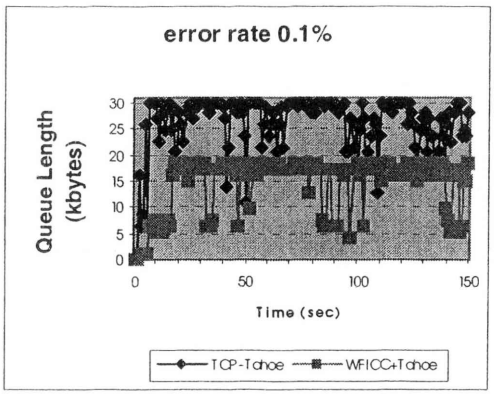


Figure 4.20 Goodput with FICC

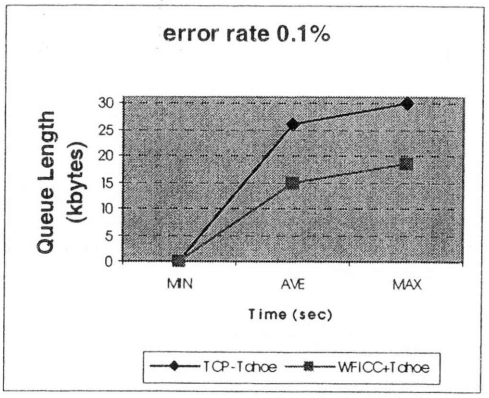
4.4 Comparisons and Evaluations

4.4.1 Queue Length and Packet Loss

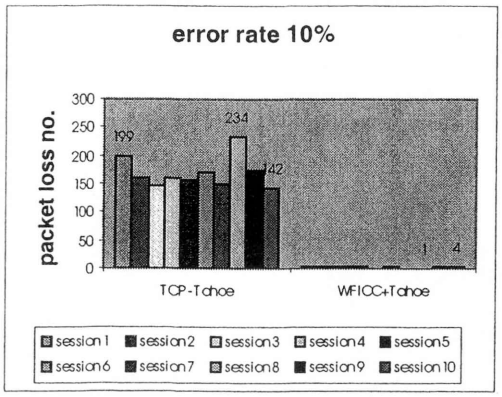
The queue lengths and packet losses of bottlenecked wireless link with regular TCP and WFICC schemes are given in Figure 4.21. Figure 4.21 shows that TCP is a passive congestion control algorithm under which congestions and packet losses occur frequently in case of traffic overloading. However, WFICC can control the queue length around the target point (In this simulation, Q_0 = 0.5 * buffersize = 15600 bytes). With FICC, there is no congestion and no packet loss due to congestion during the transmission. The few losses occurred due to wireless link errors. Figure 4.21 (b) gives the maximum, minimum and average queue lengths during the transmission period for two schemes. The queue length variation and average queue length with WFICC are smaller than those under regular TCP. We also run the simulations for different error rates. The results are very similar but the variations increase with WFICC as the error rate increases. This is because high error rate may cause the losses of RD control packet as well as data packets and thus affects the control of queue length.



(a)



(b)



(C)

Figure 4.21 Queue length and packet losses

4.4.2. Delay

Figure 4.22 provides the end-to-end delays for two different schemes. The results in our simulations show that the average end-to-end delay and delay variation with WFICC are much smaller than those with regular TCP.

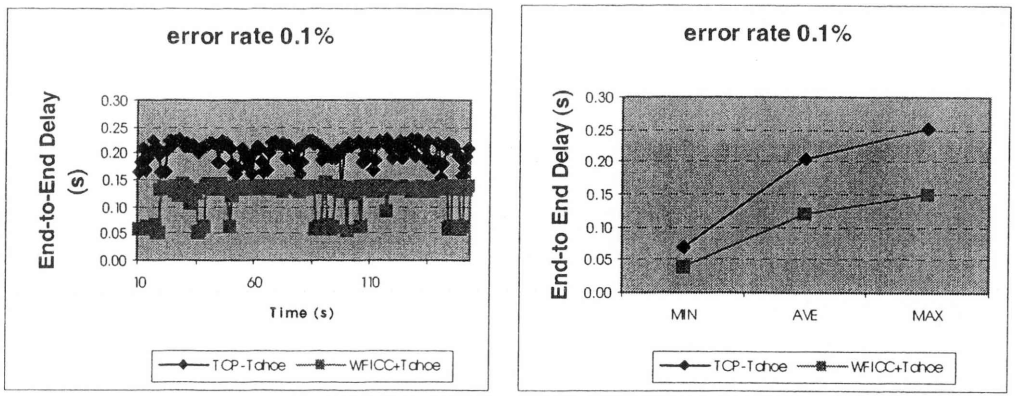


Figure 4.22 End-to-end delay

4.4.3 Goodput

Figure 4.23 provides both the throughput and goodput under TCP-Tahoe and WFICC algorithms for different packet error rates. It is shown that the bandwidth under WFICC can be used more efficiently. On average, the goodput with WFICC improves 2.5% than its counterpart. The inefficiency of TCP is because it does not have control mechanism for queue length and buffer overflow and underflow happens frequently. It should be noted that the goodput calculated with WFICC does not count the transmission of any RD packet.

WFICC improves TCP throughput by 2.5%. However, the great benefit gained here is emphasised in the fairness. One way used to measure the fairness is using index such as Raj Jain's fairness indices in which variance of session rates and mean, minimum and maximum session rate as are used as quantitative metrics. Another common method used to measure the fairness in this thesis is using the average throughput/goodput for each flow at the same specified period of time ($t_1 - t_2$) in the same condition (PER). We will see in the case of WFICC, all flows have quite similar throughput. In other words, fairness among competing flows is greatly achieved. We will discuss the details fairness gained in next section (4.4.4 Fairness)

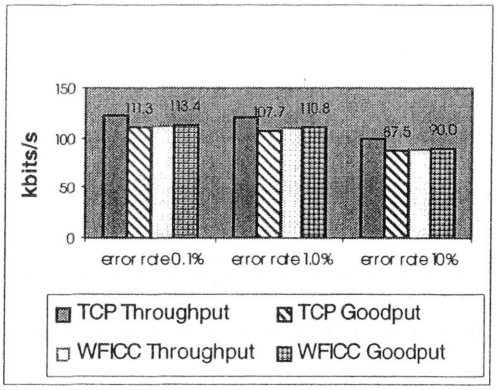


Figure 4.23 Throughputs and goodputs for different error rates

Through running simulations for different RD rates, we observed that there is a tradeoff between queue length variation and goodput. If we increase the RD packet sending rate, the queue length can be controlled better, however, the goodput will drop.

4.4.4 Fairness

Figure 4.24 provides the goodputs of 10 sessions at 0.1% error rate of wireless link with two different schemes. In Figure 4.24, with regular TCP, the goodputs of 10 sessions are unfairly allocated; they vary from 90.6 kbits/s (session 1) to 129.4 kbits/s (session 9). However, with WFICC, the goodputs of 10 sessions vary only from 113.2 kbits/s (session 2 and 5) to 114.0 kbits/s (session 8). In other words, WFICC achieves its fairness by allocating about the same amount of bandwidth to all contending connections.

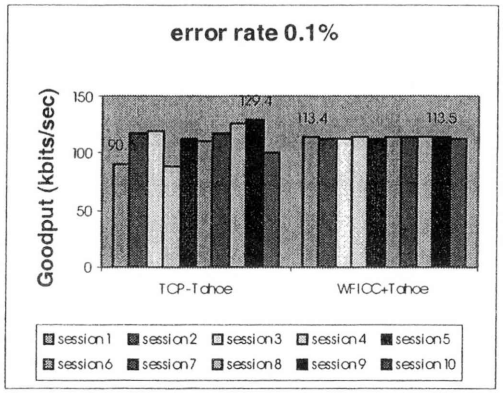


Figure 4.24 Goodputs of different sessions

This feature of WFICC comes from its design mechanisms. WFICC estimate accurately the fair share for each session at each router and convey the information to sender by RD and ACK packets.

In TCP, the detection of packet loss (duplicated acknowledgements or RTO) will trigger both retransmission and reduction of window size. In WFICC, we separate the function of packet retransmission from the reduction of window size. That is, the window size is controlled only by the traffic status at the bottlenecked link while the detection of packet loss will only trigger retransmission of packet. Through adjusting the window size or controlling the data rate, the queue length can be controlled close to its target point, namely no buffer overflow or underflow, such that the router can operate at its highest capacity and most efficient throughput can be achieved. The second step following this study will add the function to detect the packet loss over wireless link and retransmit the lost packet as early as possible. We may combine our feedback control algorithm with some methods in link layer protocols such as Snoop Protocol and Explicit Loss Notification.

The future work will also include improvements of the current feedback congestion control scheme. First, admission control/Explicit Window Adaptation (EWA) at the edge routers or other control methods will be introduced to make the control of queue length more stable and less response time. Second, WFICC/FICC will be jointly used with the other TCP variants (New-Reno, SACK).

In addition, we will exam our scheme in different network environments and traffic topologies. For example, we will test the performance in the case of different transmission direction (from mobile host to fixed or another mobile host) and the performance for multiple wireless hops.

4.5 Summary

This chapter proposed a scheme WFICC to improve TCP performance in wired-cum-wireless networks and presented some simulation results. WFICC jointly works with regular TCP to control the window sizes by the network congestion status. By employing WFICC scheme, network congestion and the data losses due to the congestion can be avoided. The variations in queue length and thus in queuing delay are significantly reduced. The sessions in the

802.11 LAN can share the bandwidth fairly. The experiments also show the gain in the effective throughput when traffic load is heavy in the network.

Chapter 5

Window-based Fair Intelligent Congestion Control (WFICC) Algorithm with Explicit Lost Notification (ELN) Deployment

5.1. Introduction

Wireless Fair Intelligent Congestion Control (WFICC) protocol has been designed by Advanced Research in Networking (ARN) group to provide goodput improvement and fairness in wired-cum-wireless networks. One disadvantage of WFICC is that the TCP source still does not know whether the end to end losses due to congestion or errors.

Explicit Loss Notification (ELN) algorithm is designed at Base Station (BS). On the forward path, BS detects congestion related losses by examining out-of-order arriving packets and stores in a Hole list. On the backward path, when the BS receives duplicate ACK, it checks sequence number of an ACK to see if there exists any correspondence in the hole list, if not

ELN flag in TCP header of the ACK is set to 1, so that the sources realize error-related losses and does not reduce the congestion window. The TCP sender retransmits the lost packet without reducing window resulting better performance.

Wireless Fair Intelligent Congestion Control with ELN (WFICC-ELN) is a combination of WFICC and ELN algorithms which give better performance and fairer goodput compared to its counterparts TCP and WFICC in the wired-cum-wireless networks.

In this chapter, we combine WFICC with ELN algorithm which gives extra information on top of WFICC by setting ELN flag in the TCP ACK header so that the sender can distinguish the lost packets occur due to either congestion or error rate and decides whether or not reducing congestion window.

5.2 New Explicit Loss Notification Algorithm Design

Our scheme is designed for wired-cum-wireless environments, which is a combination of an improved ELN Scheme [13] and the WFICC developed for wireless networks in chapter 4. We first state the main idea of the ELN scheme, then combine with our WFICC algorithm. A detailed prescription of the procedures in Base Station and the Mobile Host is presented.

Based on the setup in Figure 5.4, the bandwidths are assigned to all links so that no congestion occurs. In order to do so, link between source and e1 has 0.2Mbps, link between e1 and c1 has 1Mbps and bandwidth between c1 and e2 is 1Mbps; bandwidth between e2 and destination is around 1.35Mbps (IEEE 802.11 has 2Mbps for raw data). It can be assumed that all losses now occur by errors on wireless link.

At Base Station

FORWARD

Upon receiving data packet, the BS compares the sequence number with the highest sequence received packet.

All packets arrive at BS are classified into three classes: *retransmit packets*, *new packets* in the normal TCP sequence, and *out-of-order packets*. A retransmit packet is a packet that may be sent before, but is retransmitted due to packet loss. A new packet in the normal TCP sequence is a packet in the normal increasing sequence arriving at the destination and this is a common case. An out-of-order packet is a packet that is not in the normal increasing sequence arriving at the BS due to the packet losses over either wired or wireless link.

All packets are classified by checking the serial number (SN). Let n be the SN of the most recently received packet and I be the SN of the arriving packet

1. If i > (n + 1), the arriving packet is an out-of-order packet
2. If i = (n + 1), the arriving packet is a new packet in the normal TCP sequence
3. If i < (n + 1), the arriving packet is a retransmit packet.

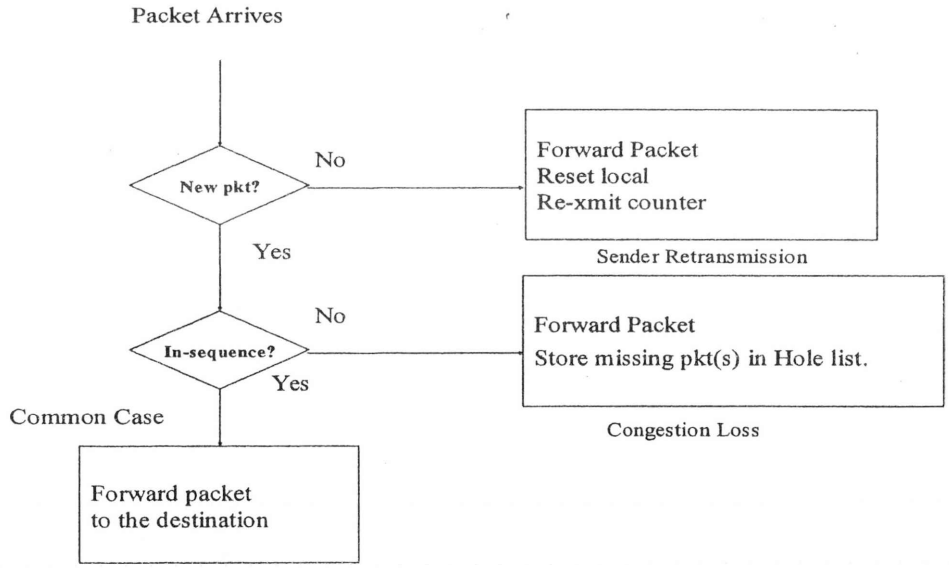


Figure 5.1 Flowchart for TCP packets at Base Station

When a new packet in the normal TCP sequence arrives at the BS, the sequence number of the packet is recorded and it is forwarded to the destination (MH). When an out-of-order packet arrives at the BS, the missing packet sequence number is recorded to the hole list.

BACKWARD:

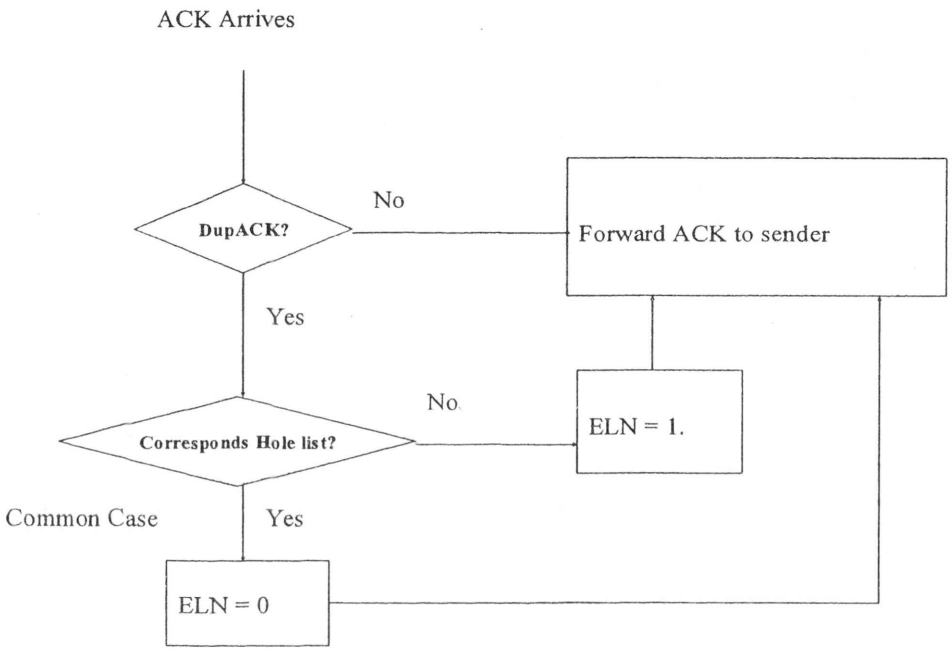


Figure 5.2 Flowchart for ACK packets at Base Station

On the backward path, the BS the ACK sequence with the hole list. If the correspondence exists, the ELN flag in ACK is remains 0, otherwise 1.

The reason for set ELN flag in the ACK is based on the well-known fact that no out-of-order packet exists in the LAN environment. Thus the BS can detect the packet loss precisely.

At Source

In common case, when an ACK packet arrives the source, it is checked if ELN flag is set in the header. If the ACK has ELN information in it (ELN=1), the source retransmits the requested packet immediately without invoking congestion control. If an ACK does not have ELN flag set, it is taken as usual.

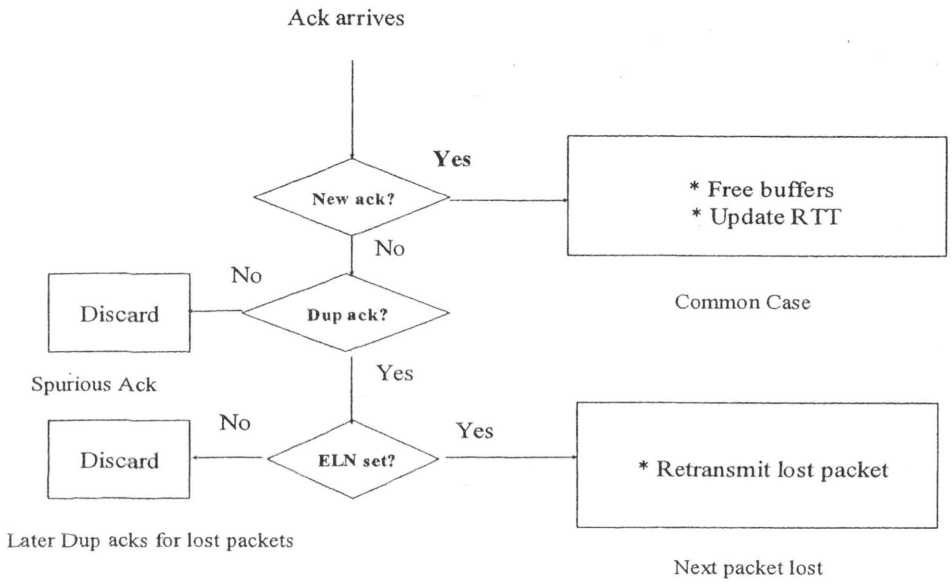


Figure 5.3 Flowchart for ACK packets at Sender

5.3 Simulation Setup for WFICC+ELN

We used ns2 [2] network simulator to evaluate WFICC-ELN protocol. Several agents are designed in C++ to implement WFICC in ns2 environment. The simulation topology is shown in Figure 5.4.

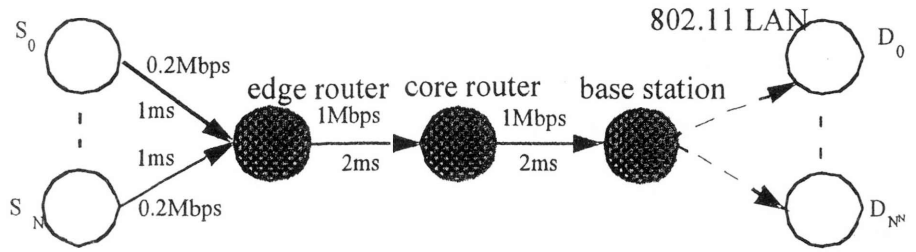


Figure 5.4 A simulation model

The bandwidths and propagation delays of wired links are given in Figure 5.4. The TCP packet size is 500 bytes and the RD packet size is 50 bytes. The buffer sizes at routers and base station are 100 packets or 50000 bytes. The simulation is run for 200 seconds. The TCP receiver window size is 50 TCP packets. The TCP version used in our simulations is TCP Tahoe. We use Two-state Error Model to generate errors on the wireless links. We assume the wired links are error free. The RD packet rate is 500. That is, there is roughly 1 RD packet generated at every 25 TCP packets.

In order to measure the performance of the protocols under controlled conditions, we generate errors on the lossy link using a two-state error model, which includes error-free and error state. Moreover, each state had a random variable determining the length of each state. We investigate the performance of the four scenarios across a range of packet loss rates from 0.01% to 100%. At high error rates, there are several occasions when multiple packets were lost within a window or when both data and retransmission data were lost. In our simulations, packet losses are due to congestion or error bit occur in different links.

5.4 Simulation Results

In this section, we provide the simulation results for WFICC-ELN and TCP for the purpose of performance comparison. The performance is evaluated in terms of queue length, congestion window, goodput and fairness. The *goodput* is defined as the number of bits of

TCP packets transmitting from the source and successfully received at the destination divided by the duration of the transmission.

We classify the simulation results into 4 categories, the first 2 cases have no errors with and without congestion, the last 2 cases have errors with and without congestion.

5.4.1 Wireless Networks with no congestion and no errors

This is considered the "ideal" situation where no congestion and errors occur. Therefore, WFICC and ELN do not have any impact in the improvement of the goodput. In the case of WFICC involved, more overheads due to RD packet generations occupy more bandwidth.

5.4.2 Wireless Networks with congestion and no errors

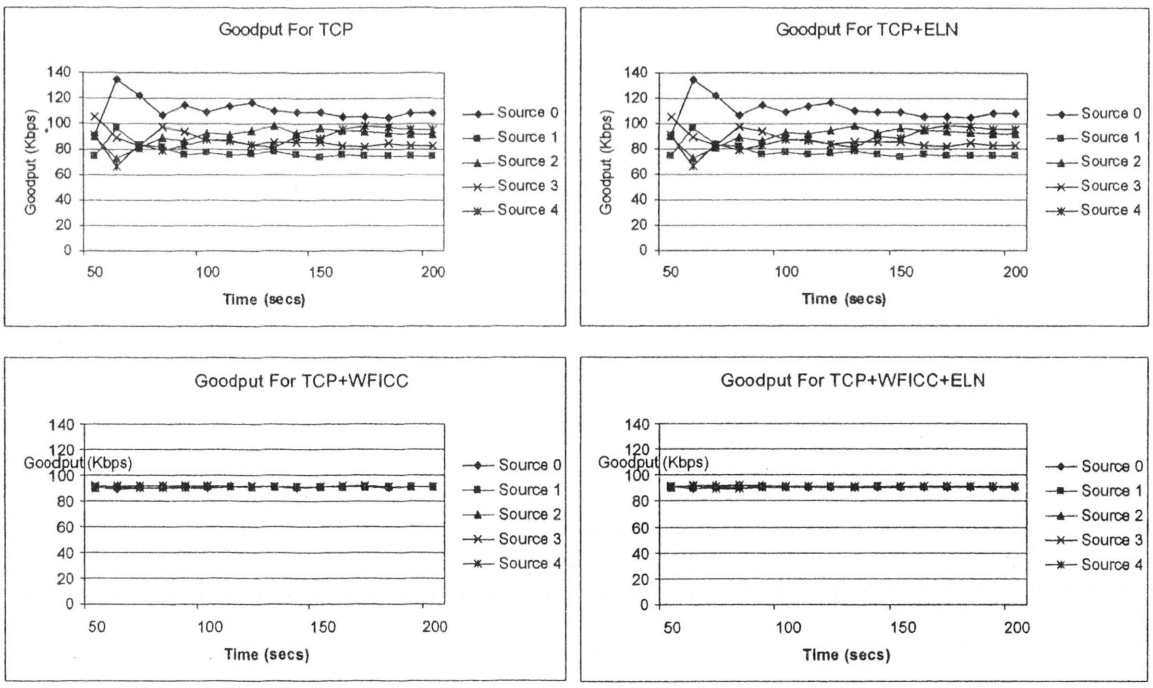


Figure 5.5 Goodput Comparison

This is similar to the wired networks where there is only congestion and errors are very low and can be ignored. WFICC gets more effective when congestion exists. As in figure 5.5 above, no congestion occurs, the goodput of wireless network with WFICC is less than its counterpart without WFICC. It is because in case if WFICC added, more RD packets occupy the bandwidth causing the throughput degrades. Now, we reduce the bandwidth between C1 and E2 from 1Mbps to 0.5Mbps, WFICC shows the effectiveness tremendously by time as shown in figure 5.6.

5.4.3 Wireless Networks with no congestion and errors

In this scenario, there is no bottleneck at any path in the network. We can see that WFICC+ELN does not pretty much outperform ELN algorithm since WFICC only works well in the case of the existence of congestion.

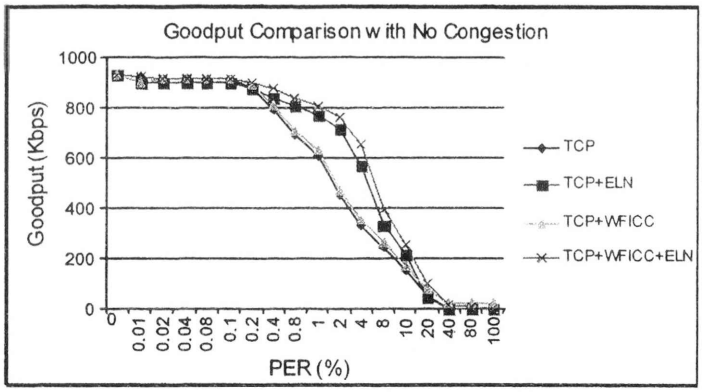


Figure 5.6 Average Goodput Comparison at different PERs

5.4.4 Wireless Networks with congestion and errors

We use two-state error model to create errors on wireless link (between E2 and receivers), with PER = 0% (no errors) PER = 0.1%, 0.2%, 0.4%, 0.8%, 1%, 2%, 4%, 8%, 10%, 20%, 40%, 80% and 100%.

- **Queue Length**

The queue lengths of bottlenecked wireless link with regular TCP, TCP+ELN, TCP+WFICC and TCP+WFICC+ELN schemes are given in Figure 5.7. Figure 5.7 shows that TCP is a passive congestion control algorithm under which congestions and packet losses occur frequently in case of traffic overloading.

However, WFICC can control the queue length around the target point (In this simulation, Q_0 = 0.5 * buffersize = 25000 bytes). With WFICC+ELN, there is no congestion and no packet loss due to congestion during the transmission. The few losses occurred due to wireless link errors but it is retransmitted by the sender without invoking window reduction. The queue length variation and average queue length with WFICC are smaller than those under regular TCP. The variation in the case of TCP+WFICC+ELN is even smaller since window reduction caused by packet loss due to error. We also run the simulations for different error rates. The results are very similar but the variations increase with WFICC as the error rate increases. This is because high error rate may cause the losses of RD control packet as well as data packets and thus affects the control of queue length.

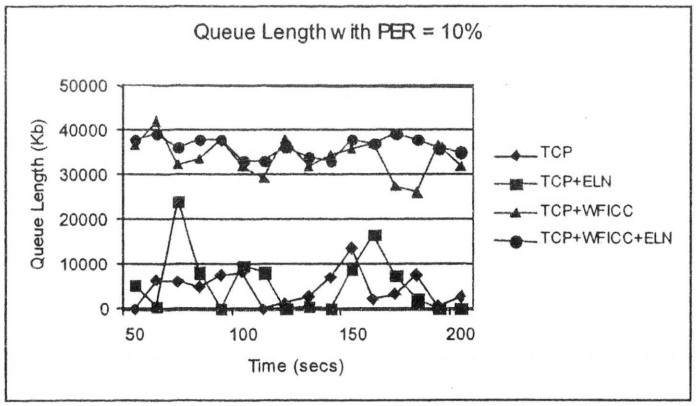


Figure 5.7 Queue length Comparison

- **Window**

With PER = 10% in TCP case, the sender is misinterpreted by the packet losses due to congestion, therefore it drops the congestion window size more frequently while in

TCP+ELN case, all window reductions due to errors are eliminated. In the cases of WFICC is involved, the congestion window is remained relatively constant, and it is even much better with combination of WFICC and ELN algorithms. It is because WFICC+ELN controls the queue length pretty well and also informs the sender if the loss is due to error so that it can immediately retransmits lost packet without reducing windows.

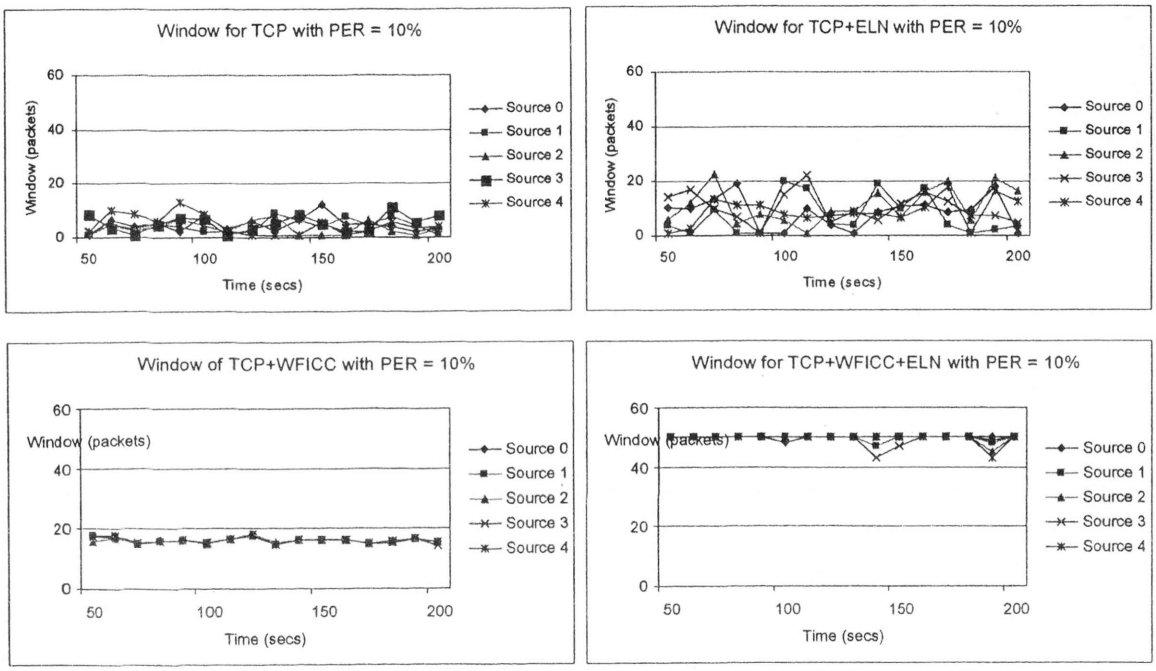


Figure 5.8 Window Comparison

- **Goodput**

Figure 5.9 provides goodputs at time = 200 secs under TCP, TCP+ELN, TCP+WFICC and TCL+WFICC+ELN algorithms for different packet error rates. It is shown that the bandwidth under TCP+WFICC+ELN can be used more efficiently. On average, the goodput with TCP+WFICC+ELN improves 5.5% than TCP alone. The inefficiency of TCP is that it does not have control mechanism for queue length and buffer overflow and underflow happens frequently and al it does not inform the sender if the packet drops due to congestion

or error, the sender drops window size, resulting in the goodput degradation. It should be noted that the goodput calculated with WFICC and WFICC+ELN does not count the transmission of any RD packet.

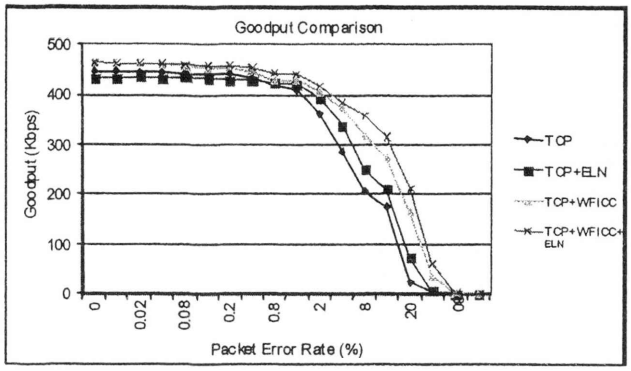


Figure 5.9 Goodputs for different error rates

Through running simulations for different RD rates, we observed that there is a tradeoff between queue length variation and goodput. If we increase the RD packet sending rate, the queue length can be controlled better, however, the goodput will drop.

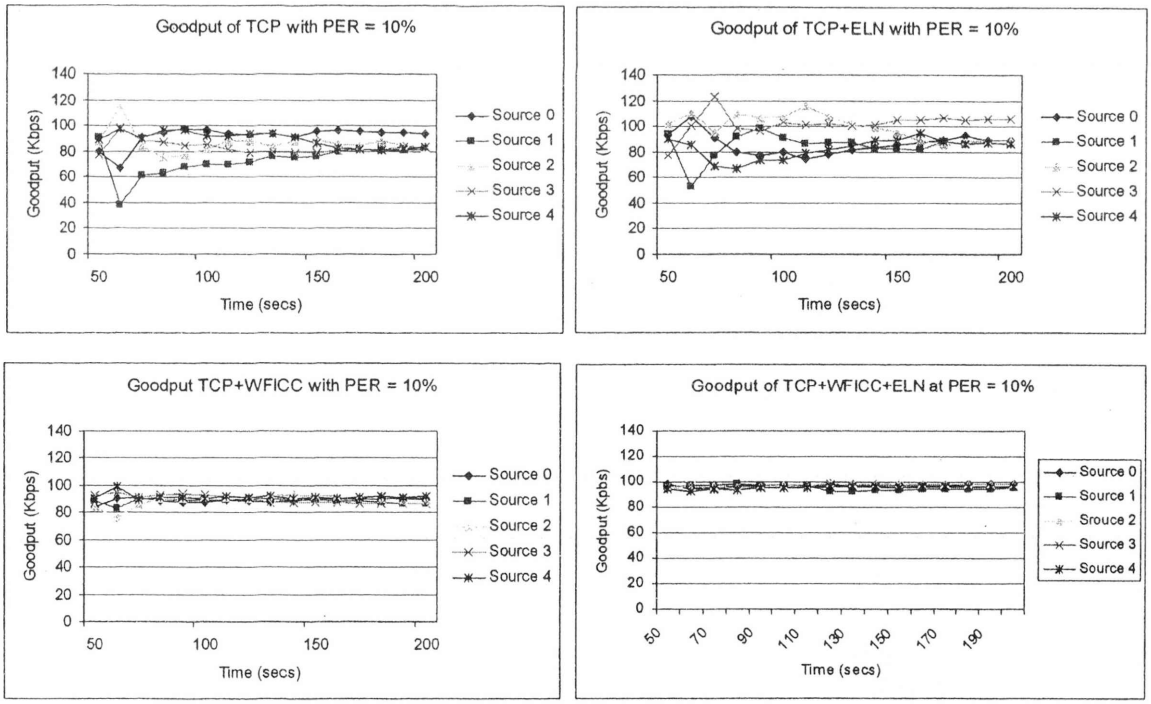


Figure 5.10 Goodput comparison at PER = 10%

- **Fairness**

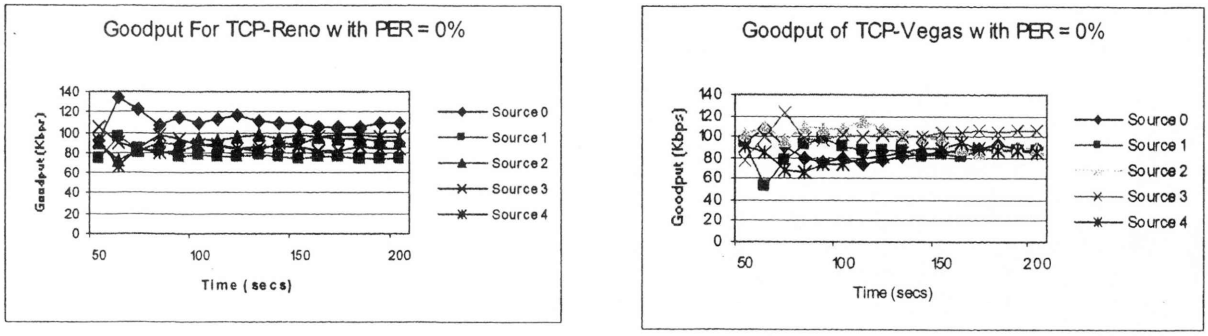
Figure 5.10 above indicates at PER = 10% the goodputs in TCP case are unfairly allocated; they vary from 81.5 Kbps (source 1) to 93.7 Kbps (source 0). In the case if WFICC-ELN, the variation between the highest (source 2: 94.6Kbps) and the lowest (source 1: 98Kbps) This feature of WFICC-ELN comes from its design mechanisms. WFICC-ELN estimates accurately the fair share for each session at each router and conveys the information to sender by RD and ACK packets combined with ELN algorithm resulting similar amount of bandwidth allocated for each contending connection.

5.4.5 Comparisons of TCP-Reno and TCP-Vegas

According to the past research, TCP Vegas does lead to a fair allocation of bandwidth and does not suffer from the delay bias as TCP Reno does through both analysis and simulations in [56]. TCP Vegas achieves better performance than TCP Reno since its bandwidth estimation does not rely on packet losses in order to estimate the available bandwidth in the network. However, when competing with other TCP Reno connections, TCP Vegas gets penalized due to the aggressive nature of TCP Reno.

TCP Vegas enhances the congestion avoidance algorithm of TCP Reno [54]. In essence, TCP Vegas dynamically increases/decreases its sending window size according to observed RTTs (Round Trip Times) of sending packets, whereas TCP Tahoe/Reno continues increasing its window size until packet loss is detected. The authors in [55] conclude through simulation and implementation experiments that TCP Vegas can obtain even better throughput than TCP Reno.

Simulations in Figure 5.11 shows that in the condition of no error, TCP-Vegas outperforms its counterpart TCP-Reno in achieving the better goodput and especially the fairness among contending sources. At time t = 200 secs, in figure 5.11 (a), the difference between the highest goodput (Source 0 = 114Kbps) and lowest goodput (Source 1 = 75Kbps) is 39Kbps while difference in Figure 5.11 (b) between the highest goodput (Source 3 = 108 Kbps) and lowest goodput (Source 2 = 87 Kbps) is 21 Kbps.

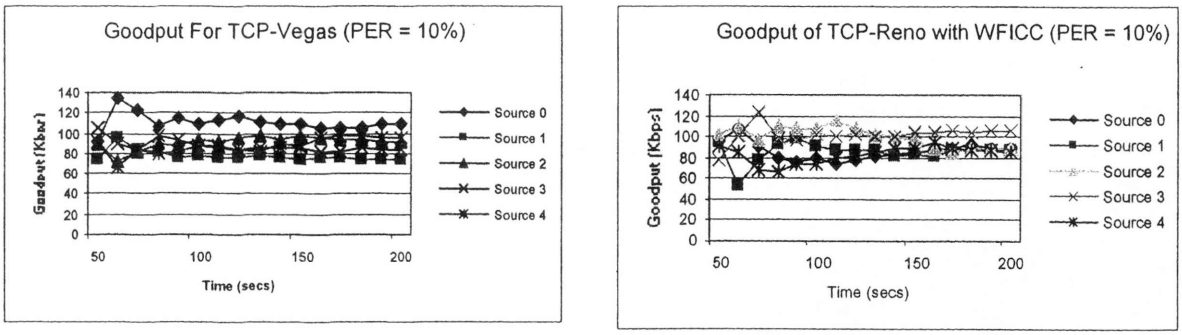


(a) (b)

Figure 5.11 TCP-Reno and TCP-Vegas Comparison at PER = 0%

It is well-known that the approach using TCP-Vegas and Random Early Detection (RED) achieves good fairness but when the available buffering is limited, it becomes ineffective [53].

On the other hand, Wireless Fair Intelligent Congestion Control (WFICC) investigates relationship between packet loss and network congestion and introduces a feedback based end-to-end congestion control algorithm to the wireless network. It tries to maintain the queue length at the bottlenecked router close to a target point to avoid router buffer overflow and underflow. Thus it can avoid congestions and minimize delay variations. It also attempts to allocate the available bandwidth fairly among the sessions competing for the same output link, resulting fair bandwidth allocation to all sessions.



(a) (b)

Figure 5.12 TCP-Vegas and TCP-Reno with WFICC at PER = 10%

Figure 5.12 (b) shows that TCP-Reno with WFICC gives better fairness than TCP-Vegas. Let have a look at Figure 5.12 (a), at time t = 150 secs, the highest goodput (Source 0) is 115 Kbps and the lowest goodput (Source 1) is 70 Kbps, therefore the difference between the highest and lowest goodputs is 45Kbps while in Figure 5.12 (b), the highest goodput (Source 3) is 105 Kpbs and the lowest goodput (Source 1) is 85 Kbps, so the difference is 20 Kpbs which is smaller than in the case of Figure 5.12 (a).

Let take another instant time t = 200 secs, in Figure 5.12 (a), the difference between the highest (Source 0 = 110 Kbps) and the lowest (Source 1 = 73 Kbps) goodputs is 37 Kbps, and in Figure 5.12 (b), the difference is 12 Kbps (the highest goodput Source 3 = 102 Kbps and the lowest goodput Source 2 is 90 Kbps) which indicates that the fairness is successfully achieved in the case of TCP-Reno with WFICC in comparison with TCP-Vegas itself.

In this thesis, TCP Reno is used for most simulations since TCP Reno is widely used by many Internet services including HTTP and FTP.

5.5 Summary

In TCP, the detection of packet loss (duplicated acknowledgements or RTO) will trigger both retransmission and reduction of window size. In WFICC and WFICC+ELN, we separate the function of packet retransmission from the reduction of window size. That is, the window size is controlled only by the traffic status at the bottlenecked link while the detection of packet loss will only trigger retransmission of packet. Through adjusting the window size or controlling the data rate, the queue length can be controlled close to its target point, namely no buffer overflow or underflow, such that the router can operate at its highest capacity and most efficient throughput can be achieved. The paper adds ELN algorithm to detect the packet loss over wireless link and retransmit the lost packet as early as possible.

In addition, we will examine our scheme in different network environments and traffic topologies. For example, we will test the performance in the case of different transmission direction (from mobile host to fixed or another mobile host) and the performance for multiple wireless hops.

5.6 Conclusions

This chapter proposed a scheme of WFICC+ELN to improve TCP performance in wired-cum-wireless networks and presented some simulation results. WFICC+ELN jointly works with regular TCP to control the window sizes by the network congestion status and informs the sender if the packet losses due to error so that the sender will not reduce window size and retransmit the lost packet immediately. By employing WFICC+ELN scheme, network congestion, the data losses due to the congestion and window reduction can be significantly reduced (Figure 5.8). The variations in queue length and thus in queuing delay were also significantly reduced (Figure 5.7). The sources in the IEEE 802.11 LAN can share the bandwidth more fairly (Figure 5.10). The experiments also showed the gain in the effective goodput when traffic load was heavy in the network.

Chapter 6

Feedback-Based Congestion Control and Explicit Window Adaptation in A Hybrid Network with Wired and Wireless Links

This chapter investigates the effects of network congestion, which often happens over low bandwidth wireless link, and QoS performance (e.g. fairness, delay variation) of multiple sessions of TCP traffic in a hybrid network. We proposed a framework, which consists of two main algorithms, feedback based congestion control and explicit window adaptation. Simulations results demonstrate that by eliminating buffer overflows caused by congestions, the framework provides desirable fairness, reduces delay variation and improves effective throughput.

6.1 Introduction

As mention in Chapter 1, congestions in TCP are not acknowledged (by ACKnowledgement packets) until packet loss or timeout occurs. In addition, the duplicate ACKs do not provide any information on whether packet losses are caused due to congestions or link errors. It assumes that all packet losses are due to congestions somewhere in the network [1]. Thus whenever packet loss is detected, TCP sender retransmits the packet and shrinks its window size at the same time.

Many schemes have been proposed to address the issue. They either tried to decouple the congestion control from retransmission of lost packets or detect and retransmit the lost packets as early/closely as possible. The representative examples are Explicit Loss Notification (ELN) [13], Explicit Congestion Notification (ECN), Indirect TCP [15], and Snoop. ELN can provide the TCP sender the reasons of packet loss, but there have been no efficient solutions to implement it yet [14]. Indirect TCP decompose the TCP connection into two sub-connections for the wired and wireless parts of the path. However, this approach does not ensure end-to-end delivery of packets and does not work well if the connection is split several times over the path of the connection. Snoop protocol, a sort of link layer proposal, introduces a snoop agent at the base station. The agent monitors every packet that passes in both direction and maintains a cache of TCP segments that have not yet been acknowledged by the receiver. It detects packet loss by duplicate ACKs or a local Retransmission TimeOut (RTO) and retransmits the lost packet. However, snoop protocol like any other link layer methods cannot shield all the packet losses. In addition, it does not work well in certain wireless environments where a fair RTO is difficult to calculate [18].

Different from the previous research work, the research work investigates the effects of network congestion and solves the deficiency of TCP in misinterpreting the link-related packet loss. It is focused on improving Quality of Service (QoS) performance of TCP for multiple sessions in a hybrid network. A framework is proposed which consists of the following two components:

1) Feedback congestion control. We use Fair Intelligent Congestion Control (FICC) algorithm [1, 19] as the feedback congestion control algorithm. FICC continuously collects the traffic information within the network and calculates the fair share for individual sessions based on the information.

2) Explicit window adaptation. An explicit window adaptation algorithm calculates explicit window size corresponding to the fair share for individual sessions. The explicit window size is updated and provided to the TCP sender continuously.

We implement the framework in Network Simulator Version 2 (NS2) [13]. The performance was examined in a simple network scenario: several fixed hosts send TCP traffic to mobile

hosts at IEEE 802.11 LAN. The simulation results show that the framework can successfully avoid congestion, achieve desirable fairness, reduce delay variations and improve the effective throughput in the hybrid network. The framework proposed is applicable to other hybrid network scenarios and traffic other than TCP. It also can be extended to jointly use with DiffServ architecture designed by IETF.

6.2 Hybrid Network Model and Design Architecture

An example of a hybrid network is given in Figure 6.1. We assume that there are a number of n fixed hosts (FHs) sending TCP data to a number of n mobile hosts (MHs), respectively. Namely, there are a number of n communication pairs. Without loss of generality, suppose the TCP packets traverse a source edge router (SER), a core router (CR) and a base station (BS) from FHs to the MHs.

We proposed a framework that consists of two key components, namely, feedback congestion control and explicit window adaptation. The combination of two components will avoid congestion, achieve fair and efficient use of network bandwidth, and naturally decouple the congestion control from retransmission of lost packets.

Fair Intelligent Congestion Control (FICC) algorithm [1, 19] is used as the feedback congestion control algorithm. FICC loop from SER to BS is illustrated in Figure 1. FICC continuously send Resource Discovery (RD) packets to collects network traffic information and, based on the information, calculates the fair share for individual sessions. The fair share is continuously updated and feedbacked to the Source Edge Router (SER).

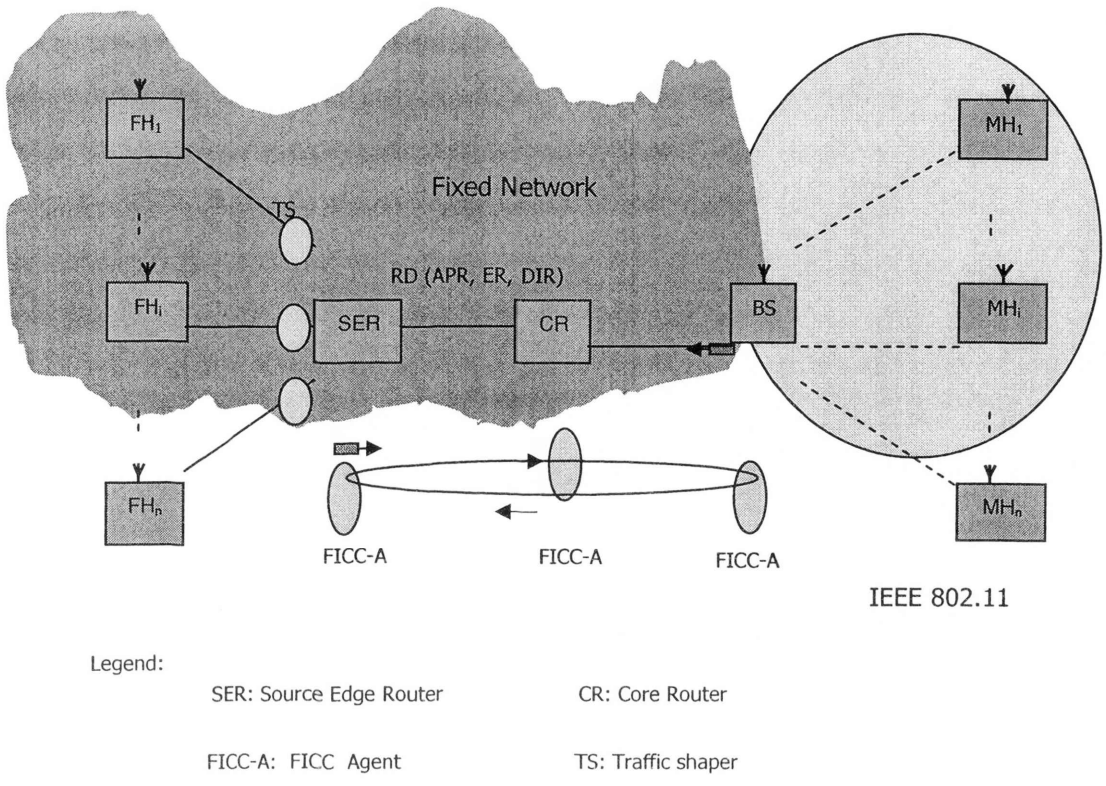


Figure 6.1 Illustration of a simple hybrid network and the proposed framework

Fair Intelligent Congestion Control (FICC) algorithm [1, 19] is used as the feedback congestion control algorithm. FICC loop from SER to BS is illustrated in Figure 6.1. FICC continuously send Resource Discovery (RD) packets to collects network traffic information and, based on the information, calculates the fair share for individual sessions. The fair share is continuously updated and feedbacked to the Source Edge Router (SER).

Also shown in Figure 6.1, each session has a traffic shaper (TS) at the entrance of SER. The shaper contains an explicit window adaptation (EWA) node, a token bucket and a small shaping buffer. The token bucket keeps sending the traffic at the rate of fair share given by FICC. An EWA algorithm at EWA node calculates the explicit window size (EWS) corresponding to the buffer fill level. The EWS is continuously updated and feedbacked to the TCP sender by ACK packet. EWS is assigned as receiver window size at ACK packet

header. Therefore, the buffer fill will be maintained at certain level such that the buffer will never go overflows and underflows. The algorithm details of EWA are is introduced in Section 6.3.

6.3 Explicit Window Adaptation Algorithm

An EWA algorithm is the other critical component to be used in combination with congestion control algorithm. It transforms the fair share ER to a matching window size EWS for TCP sender. EWS is delivered by ACK packet and regarded as receiver window size at TCP sender. The EWA is implemented at traffic shaper, which is illustrated in Figure 6.2.

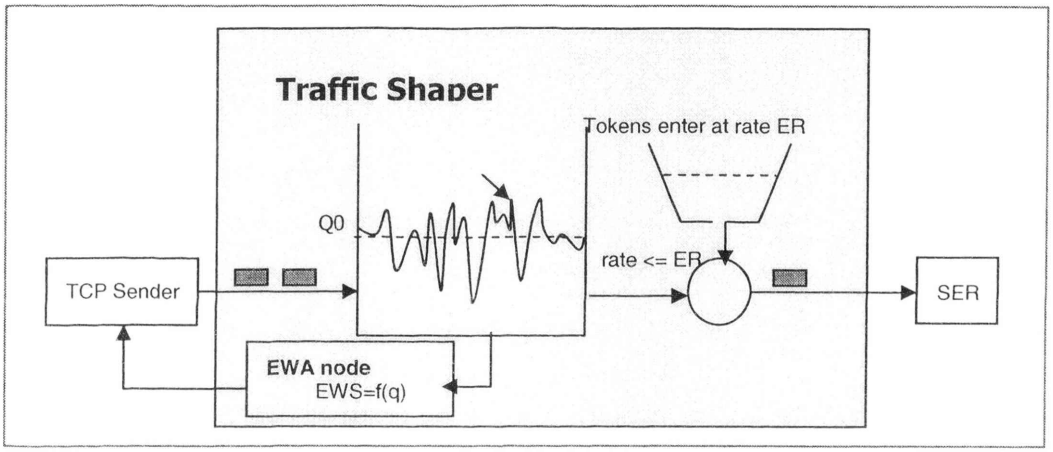


Figure 6.2 A traffic shaper

The traffic shaper consists of a token bucket and EWA node. The token bucket generates tokens at rate of ER which means TCP packets enter the network at rate ER. In this study, we design an EWA algorithm that is very similar to the control function of FICC described in the previous section. Again we select a target point Q_0 and try to keep the fill level of the shaping buffer close to the target point. The calculation details are illustrated in Figure 6.3

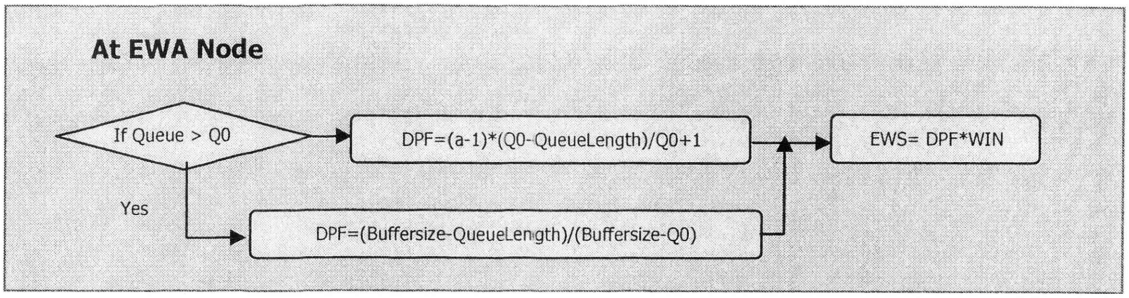


Figure 6.3 Explicit window adaptation algorithm

6.4 Simulation Setup

We used ns2 [2] network simulator to evaluate the framework. Several agents are designed in C++ to implement the schemes in ns2 environment. The simulation topology is shown in Figure 6.4.

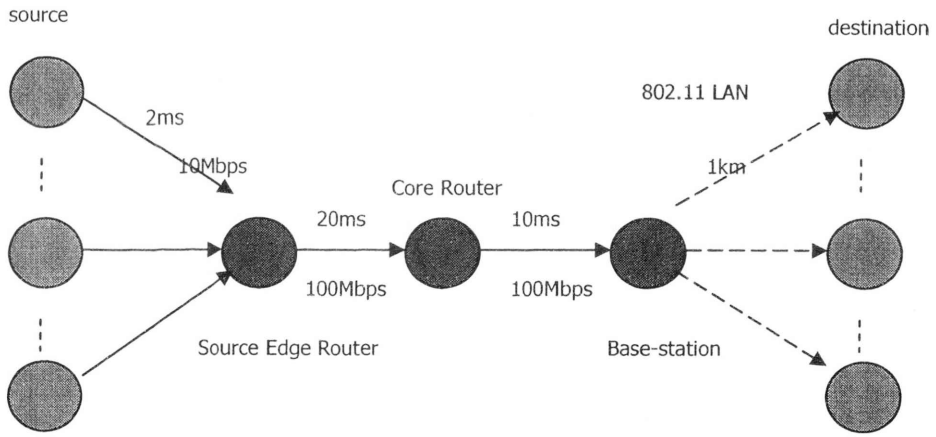


Figure 6.4 A simulation model

The bandwidths and propagation delays of wired links are given in Figure 6.4. The TCP packet size is 540 bytes and the RD packet size is 50 bytes. The buffer sizes at routers and base station are 20 packets, namely 10800 bytes. The target BUR is 0.3. The buffer sizes at traffic shapers are 10 packets, namely 5400 bytes, and the target BUR is 0.2. The simulation is run for 100 seconds. The TCP receiver window size is 50 TCP packets. The TCP version

used in our simulations is TCP Tahoe. We use uniform error model to generate errors on the wireless links. We assume the wired links are error free. The RD packet rate is 50. That is, there is roughly 1 RD packet generated at every 5 TCP packets.

6.5 Simulation Results

We provide the simulation results and evaluate the performance of FICC and EWA algorithms in the following sections.

6.5.1 Performance of FICC

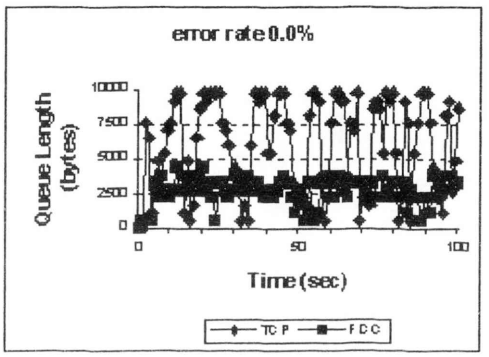
In this section, we provide the simulation results for both regular TCP and FICC and for the purpose of performance comparison. The performance of FICC is evaluated in terms of queue length, packet loss, end-to-end delay, throughput, goodput and fairness. The *throughput* is defined as the number of bits of all the TCP packets that are transmitted at the source (including RD packets if FICC is used) divided by the duration of the transmission. The *goodput* is defined as the number of bits of TCP packets that are transmitted at the source and successfully received at the destination divided by the duration of the transmission.

- **Queue Length and Packet Loss**

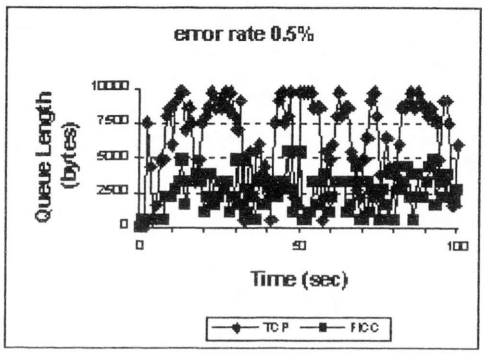
We run the simulations for different error rates. The results are quite similar. The queue lengths of bottlenecked wireless link for different packet error rates with regular TCP and FICC schemes are given in Figure 6.5. Figure 6.5 shows that TCP is a passive congestion control algorithm under which congestions and packet losses occur frequently in case of traffic overloading. However, FICC can always control the queue length around the target point (In this simulation, $Q_0 = 0.3 * \text{buffersize} = 3240$ bytes), even when the error rate is high (e.g. 5%). Table 1 gives the minimum, average and maximum queue lengths for different packet error rates when FICC algorithm is used. It shows that the queue length variation increases as the error rate becomes higher though the queue lengths are still well controlled. This is because high error rate cause more losses of data packets and thus affects the control

of queue length. The minimum, average and maximum queue lengths for packet error rate 1% with FICC algorithm is illustrated in Figure 6.6 (a). The queue length variation and average queue length with FICC are much smaller than those with regular TCP.

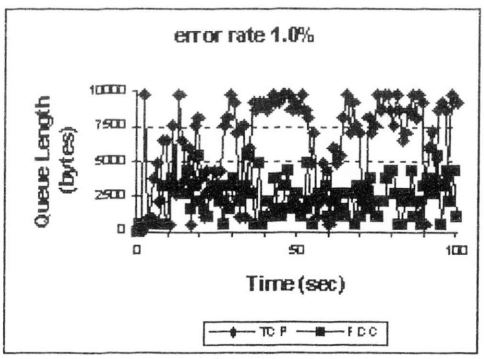
Figure 6.6 (b) illustrates the packet losses for both regular TCP and FICC when error rate is 1%. With FICC, there is no congestion and no packet loss due to congestion during the transmission. The few losses occurred due to wireless link errors and not recovered by link layer protocol.



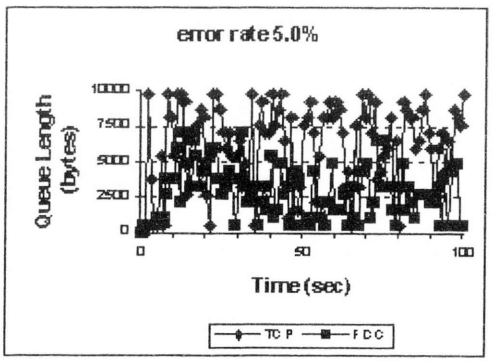
(a) Error rate 0.0%



(b) Error rate 0.5%



(c) Error rate 1.0%

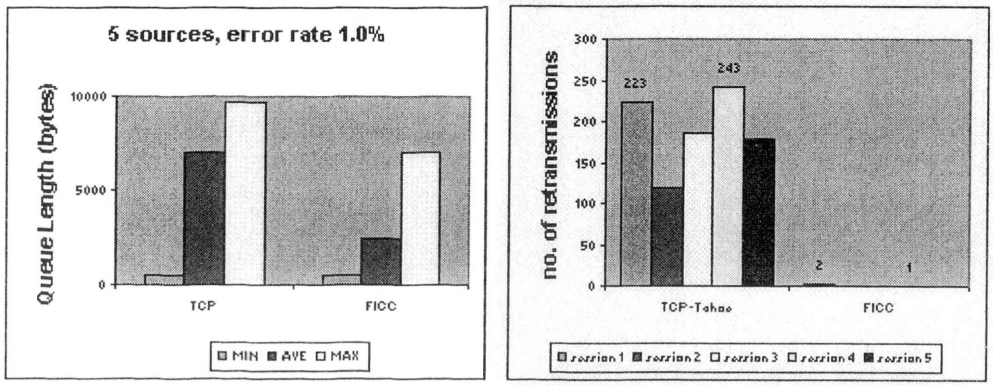


(d) Error rate 5.0%

Figure 6.5 Queue lengths vs. time at bottlenecked link for different error rates

Table 6.1 Minimum, average and maximum queue lengths for different error rates

Error Rate	0.0%		0.5%		1.0%		5.0%	
Scheme	TCP	FICC	TCP	FICC	TCP	FICC	TCP	FICC
MIN	540.0	540.0	540.0	540.0	540.0	540.0	540.0	540.0
AVE	6376.6	2723.0	6956.0	2585.9	6991.1	2479.6	6716.1	2894.2
MAX	9720.0	4320.0	9720.0	5400.0	9720.0	5400.0	9760.0	7060.0

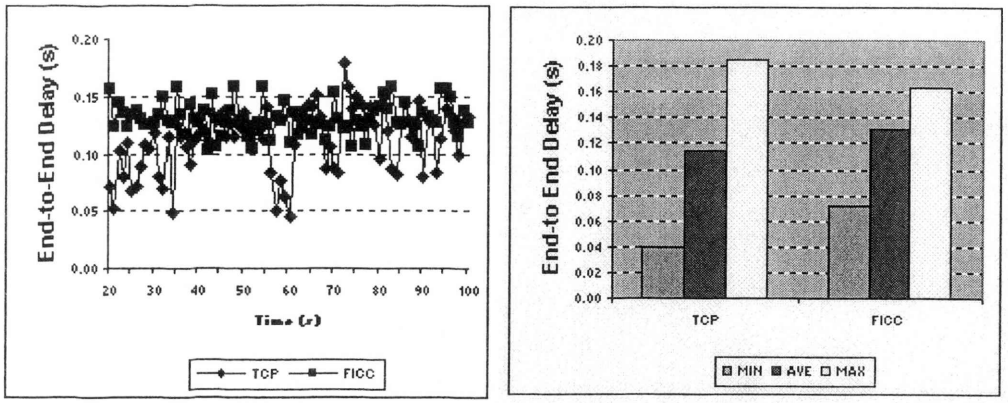


(a) Queue lengths (b) No. of lost packets

Figure 6.6 Queue lengths and packet losses at error rate 1.0%

- **Delay**

Figure 6.7 provides the end-to-end delays for two different schemes with error rate 1%. The results in our simulations show that the delay variations with FICC are smaller than those with regular TCP. It should be noted that the average delay with FICC is higher than regular TCP because the use of shaping buffer at the source end. Further study to reduce this additional delay by using one common shaping buffer is under consideration.



(a) (b)

Figure 6.7 Average end-to-end delay

- **Goodput**

Figure 6.8 and Table 6.2 provides both the throughput and goodput under regular TCP and FICC algorithms for different packet error rates. It is shown that the bandwidth under FICC can be used more efficiently. On average, the goodput with FICC improves 2.2% than its counterpart. The inefficiency of TCP is because it does not have control mechanism for queue length and buffer overflow and underflow happens frequently. It should be noted that, as defined, the goodput calculated with FICC does not count the transmission of any RD packet.

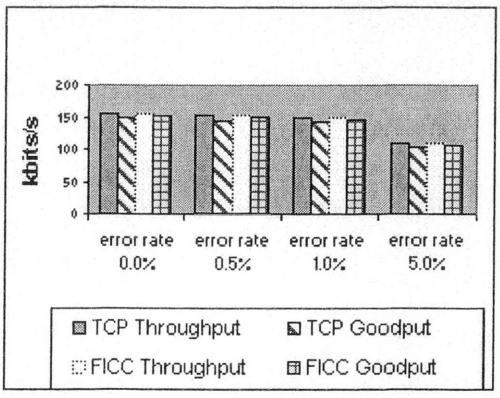


Figure 6.8 Throughputs and goodputs for different error rates

Through running simulations for different RD rates, we observed that there is a tradeoff between queue length variation and goodput. If we increase the generation frequency of RD packet, the queue length can be controlled better, however, the goodput will drop.

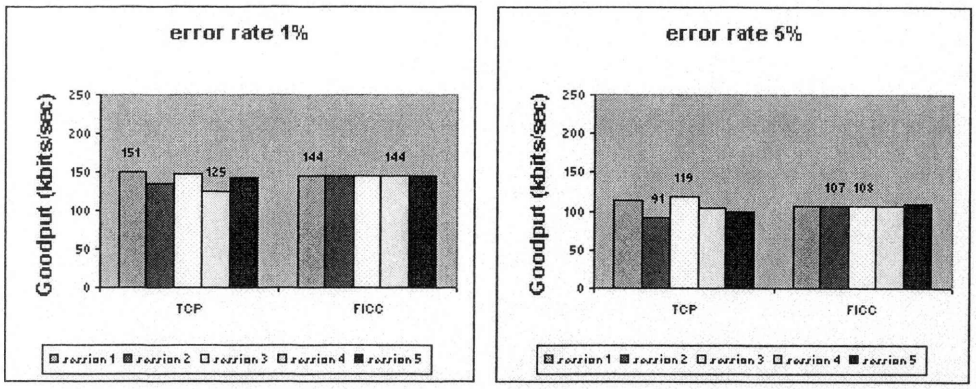
Table 6.2 Comparison of throughputs and goodputs for different error rates

Packet Error Rate		0.0%	0.5%	1.0%	5.0%
Throughput (kbits/s)	TCP	159.2	148.8	145.5	110.4
	FICC	156.1	147.2	147.3	109.7
Goodput (kbits/s)	TCP	151.5	144.3	139.8	105
	FICC	153	147.2	144.4	107.5
	Gain (%)	1.0	2.0	3.3	2.4

- **Fairness**

Figure 6.9 provides the goodputs of 5 sessions at packet error rates 1% and 5% with two different schemes. Figure 6.9 (a) shows that, with regular TCP and error rate 1%, the goodputs of 5 sessions vary from 125 kbps (session 1) to 151 kbps (session 4). The difference is about 20%. In Figure 6.9 (b) shows that, with regular TCP and error rate 5%, the goodputs vary from 91 kbps (session 2) to 119 kbps (session 3). The difference is about 30%. However, with FICC and error rate 1%, the goodputs of 5 sessions are basically the same. Even at error rate 5%, 5 sessions still takes roughly the same amount of bandwidth. This shows the unfairness of bandwidth allocation with regular TCP and the desirable fairness of bandwidth allocation with FICC.

Achieving desirable fairness is a main feature of FICC scheme, which comes from its design principle. FICC estimate accurately the fair share for each session at each router and delivery the information to sender by RD and ACK packets constantly

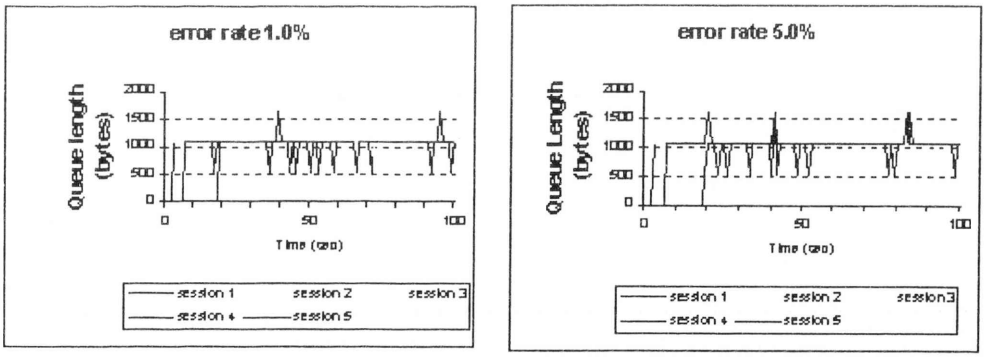


(a) Error rate 1.0% (b) Error rate 5.0%

Figure 6.9 Goodputs of different error rates

6.5.2 Performance of Explicit Window Adaptation Algorithm

In this section, we evaluate the performance of EWA algorithm in terms of queue length of shaping buffer, window size and data sending rate. Figure 6.10 provides the queue lengths of the buffers of individual sessions at different error rates. EWA algorithm can always control the queue length around the target point (In this simulation, $Q_0 = 0.2 * \text{buffersize} = 1080$ bytes), even when the error rate is high (e.g. 5%).

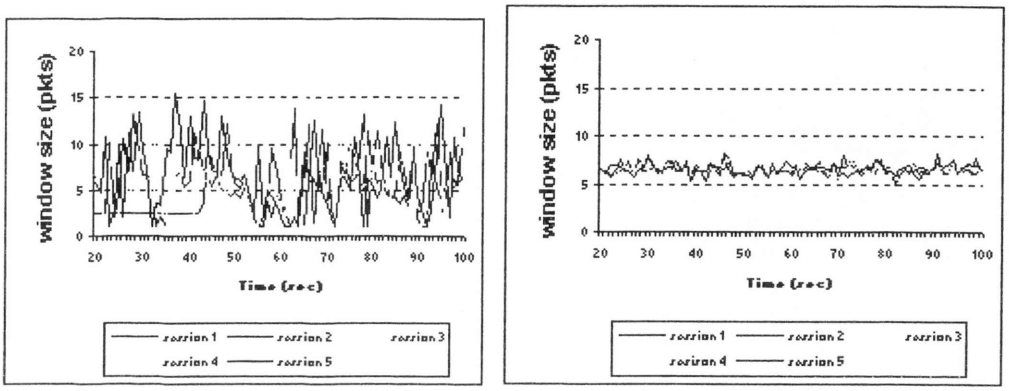


(a) Error rate 1.0% (b) Error rate 5.0%

Figure 6.10 Queue lengths of shaper buffer at different error rates

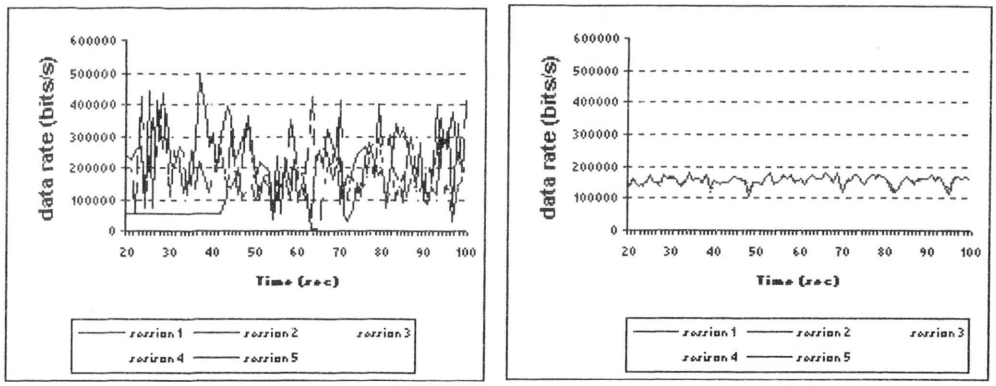
Figure 6.11 provides the window sizes of individual sessions with TCP and FICC at error rate 1%. Figure 6.12 provides the bit-sending rates of individual sessions for two schemes at

error rate 1%. From Figures 6.11 (b) and 6.12 (b), we can also see the fairness and the stability of the queue length with the proposed framework, the window size of all sessions in Figure 6.11 (b) and 6.12 (b) are varying from 6 pkts to 8 pkts resulting the fairness in goodput/throughput.



(a) Regular TCP (b) FICC

Figure 6.11 Comparison of TCP window sizes



(a) Regular TCP (b) FICC

Figure 6.12 Comparison of bit-sending rates

6.6 Summary

This chapter proposed a framework that consists of two main algorithms, namely, FICC and EWA, to improve QoS performance of TCP for multiple sessions in a hybrid network. By employing the framework, network congestion and the data losses due to the congestion can be avoided. The variations in queue length and thus in queuing delay are significantly reduced. The sessions in the 802.11 LAN can share the bandwidth fairly. The experiments also show certain gain in the effective throughput.

This framework naturally decouples network congestion control and lost packet retransmission. Following this study, we will add the functions for the framework to detect the packet loss over wireless link and retransmit the lost packet as early as possible. The future work will also include: 1). Jointly use the framework with the other TCP variants (Reno, SACK). 2). Exam the framework in different network environments and traffic topologies, for example, different transmission direction (from mobile host to fixed or another mobile host) and multiple wireless hops. 3). Extend to jointly use with DiffServ architecture at wireless networks.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, the core chapters are chapters 4, 5 and 6 which described new algorithms designed to improve throughput and fairness of TCP on wireless networks.

Chapter 4 proposed a scheme WFICC to improve TCP performance in wired-cum-wireless networks and presented some simulation results. WFICC jointly worked with regular TCP to control the window sizes by the network congestion status. By employing WFICC scheme, network congestion and the data losses due to the congestion can be avoided. The variations in queue length and thus in queuing delay are significantly reduced. The sessions in the IEEE 802.11 LAN can share the bandwidth fairly.

In TCP, the detection of packet loss (duplicate acknowledgements or RTO) triggers both retransmission and reduction of window size. In WFICC and WFICC+ELN, we separated the function of packet retransmission from the reduction of window size. That is, the window

size is controlled only by the traffic status at the bottlenecked link while the detection of packet loss will only trigger retransmission of packet. Through adjusting the window size or controlling the data rate, the queue length can be controlled close to its target point, namely no buffer overflow or underflow, such that the router can operate at its highest capacity and most efficient throughput can be achieved. The algorithm added ELN algorithm to detect the packet loss over wireless link and retransmit the lost packet as early as possible.

Chapter 5 proposed a scheme WFICC-ELN to improve TCP performance in wired-cum-wireless networks and presented some simulation results. WFICC-ELN jointly works with regular TCP to control the window sizes by the network congestion status and informs the sender if the packet losses due to error so that the sender does not reduce window size and retransmit the lost packet immediately. By employing WFICC-ELN scheme, network congestion, the data losses due to the congestion and window reduction can be avoided. The variations in queue length and thus in queuing delay are significantly reduced. The sources in the IEEE 802.11 LAN can share the bandwidth fairly. The experiments also showed the gain in the effective goodput/thoughput when traffic load is heavy in the network.

Finally, chapter 6 proposed a framework that consists of two main algorithms, namely, FICC and EWA, to improve QoS performance of TCP for multiple sessions in a hybrid network. By employing the framework, network congestion and the data losses due to the congestion can be avoided. The variations in queue length and thus in queuing delay are significantly reduced. The sessions in the 802.11 LAN can share the bandwidth fairly. The experiments also showed certain gain in the effective throughput.

7.2 Future Work

Based on chapter 4, the improvements can be made to the current feedback congestion control scheme. First, admission control/Explicit Window Adaptation (EWA) at the edge routers or other control methods will be introduced to make the control of queue length more stable and less response time. Second, WFICC/FICC will be jointly used with the other TCP

variants (New-Reno, SACK). In addition, we examine our scheme in different network environments and traffic topologies.

To continue work in chapter 5 with the above algorithms, we can further improve the performance of TCP on wireless networks. First, since the fairness is successfully achieved, we therefore can control the capability of each TCP source by employing Differentiation Mechanisms into the scheme at Mac Layer (IEEE 802.11) such as varying Backoff function, DIFS and Maximum frame length [23].

Next, we can enhance DiffServ to be well suited into wireless then embed it into WFICC, consider following factors such as Traffic Classifier and Conditioner, Signaling Mechanism, Mobility, High Loss Rate Handling, Low Bandwidth Handling, Power Constraints Handling.

We can jointly use the framework with the other TCP variants (NewReno, SACK, Vegas ...) then we can compare and find out which TCP variant achieves best performance. We will also examine the framework in different network environments and traffic topologies, for example, different transmission direction (from mobile host to fixed or another mobile host) and multiple wireless hops.

In chapter 6, the framework naturally decouples network congestion control and lost packet retransmission. Following this study, we will add the functions for the framework to detect the packet loss over wireless link and retransmit the lost packet as early as possible. The future work will also include: 1). Jointly use the framework with the other TCP variants (Reno, SACK). 2). Examine the framework in different network environments and traffic topologies, for example, different transmission direction (from mobile host to fixed or another mobile host) and multiple wireless hops. 3). Extend to jointly use with DiffServ architecture at wireless networks.

Bibliography

[1] D. B. Hoang, Q. Yu, M. Li, and D. Feng, *Fair Intelligent Congestion Control Resource Discovery Protocol on TCP Based Network*, In the Proceedings of the IFIP 6th Internetworking 2002 Symposium, October 2002.

[2] Network Simulation ns2, http://www.isi.edu/nsnam/ns

[3] J. B. Postel, *Transmission Control Protocol*, RFC, Information Sciences Institute, Marina del Rey, CA, September 1981. RFC 793.

[4] G. Xylomenos and G. Polyzos, *TCP Performance Issue over Wireless Links*, Athens Uni. of Economics and Business.

[5] P. Nhan, D. Hoang, H.Zhou and V. Merchandani, *Window-based Fair Intelligent Congestion Control with Explicit Lost Notification (ELN) Deployment*, University of Technology, Sydney. To be submitted for publication.

[6] H. Zhou, D. Hoang, P. Nhan V. Merchandani, *Feedback-Based Congestion Control and Explicit Window Adaptation in Hybrid Networks with Wired and Wireless Links*, University of Technology, Sydney. It is submitted to the IEEE Wireless Communications and Networking Conference, Atlanta, March 2003.

[7] I. Mahadevan and K. Sivalingam, *Quality of Service Architectures for Wireless Networks: IntServ and DiffServ Models*, School of Electrical Engineering and Computer Science, Washington State University.

[8] Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A., *Selective Acknowledgement options*, Internet Engineering Task Force, October 1996, RFC 2108.

[9] V. Jacobson, *Congestion Avoidance and Control*, Proceedings of the ACM SIGCOMM 88, August 1988.

[10] F. Khafizov and M. Yavuz, *Running TCP over IS-2000*, Proceedings of the IEEE International Conference on Communications, 2002, pp. 3444-3448.

[11] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen, *TCP Performance Issues over Wireless Links*, IEEE Communication Magazine, April 2001, pp. 52-58.

[12] H. Balakrishnan, V. N Padmanabhan, S. Seshan, and R. H. Kaze, "*A Comparison of Improving TCP Performance over Wireless Links*", IEEE/ACM Trans. on Networking, Vol. 5, no. 6, pp. 756-769, December 1997.

[13] H. Balakrishnan and R. H. Kaze, *Explicit Loss Notification and Wireless Web Performance*, IEEE Globecom'98, Sydney, Australia.

[14] J. Pan, J. Mark, and X. Shen, *TCP Performance and Its Improvement over Wireless Links*, Proceedings of IEEE Globecom 2000, pp. 62-66.

[15] A. Bakre and B. R. Badrinath, *I-TCP: Indirect TCP for mobile hosts*, Proceedings of the 15th Int. Conf. Distributed Computing Syst., May 1995.

[16] H. Balakrisnan, S. Seshan, and R. H. Katz, *Improving reliable transport and handoff performance in cellular wireless networks*, ACM Wireless Networks, vol. 1, Dec. 1995.

[17] C. Parsa and J. J. Gacia-Luna-Aceves, *Improving TCP performance over Wireless Networks at the Link Layer*, Mobile Networks and Applications, 1999.

[18] J. Rendon, F. Casadevall, and D. Serarols, *Snoop TCP Performance over GPRS*, Vehicular Technology Conference, 2001. . IEEE VTS 53rd, Pages: 2103-2107 vol. 3.

[19] D. B. Hoang, Z. Wang, *A Fair Intelligent Congestion Control*, Technical Report 224, ISBN 1 86487326 4, Basser Department of Computer Science, University of Sydney, Australia, 1999.

[20] D-M. Chiu and R. Jain. *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*. Computer Networks and ISDN Systems, 17:1-14, 1989.

[21] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, *TCP Vegas: New Techniques for Congestion Detection and Avoidance*, In *Proc. ACM SIGCOMM '94*, August 1994.

[22] K. Brown and S. Singh, *M-TCP: TCP for Mobile Cellular Networks*, Computer Communications Review, 27(5), October 1997.

[23] I. Aad, C. Castelluccia, *Differentiation Mechanisms for IEEE 802.11*, PLANETE project, INRIA Rhô-Alpes. ZIRST – 655, Avenue de l'Europe – Montbonnot, 38334 Saint Ismier Cedex – France.

[25] R. Caceres and L. Iftode, *Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments*, IEEE Journal on Selected Areas in Communications, 13(5), Jun 1995.

[24] S. Floyd, *TCP and Explicit Congestion Notification*, Computer Communications Review, 24(5), October 1994.

[26] S. Floyd and V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, 1(4), August 1993.

[27] T. V. Lakshman, U. Madhow, and B. Suter, *Window-based Error Recovery and Flow Control with a Slow Acknowledgement Channel: A study of TCP/IP Performance*, In

Proc. Infocom 97, April 1997.

[28] D. Lin and H. Kung, *TCP Fast Recovery Strategies: Analysis and Improvements*, In *Proc. INFOCOM*, March 1998.

[29] *OSI Transport Protocol Specification*, 1986. Standard ISO-8073.

[30] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, MA, 1994.

[31] K.K. Ramakrishnan and S. Floyd, *A Proposal to Add Explicit Congestion Notification (ECN) to IPv6 and to TCP*, Internet Draft draft-kksjf-ecn-00.txt, November 1997.

[32] W. R. Stevens, *TCP/IP Illustrated, Volume 1*, Addison-Wesley, Reading, MA, Nov 1994.

[33] H. Zimmermann, *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*, IEEE Trans. on Communications, 28(4), April 1980.

[34] D. Hoang, H. Zhou, P. Nhan and V. Mirchandani, "*Performance Study of a Feedback-Based Congestion Control Algorithm in Wireless Networks*", University of Technology, Sydney. To be submitted for publication.

[35] *Wireless Medium Access Control and Physical Layer WG, IEEE Draft Standard* P802.11, "Wirelesss LAN," IEEE Stds, Dept, D3, Jan. 1996

[36] Jason S. King, *An IEEE 802.11 Wireless LAN Security White Paper* (http://www.llnl.gov/asci/discom/ucrl-id-147478.html), October 22, 2001

[37] Brian P. Crow, Indra Widjaja, Jeong Geun Kim & Prescott T. Sakai, *IEEE 802.11 Wireless Local Area Netoworks.*

[38] Chetan Sharma, *Wireless Internet Enterprise Applications – A Wiley Tech Brief*, Wiley Computing Publishing, http://www.wiley.com/compbooks/

[39] D-M. Chiu and R. Jain, *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*, Computer Networks and ISDN Systems, 17:1-14, 1989.

[40] J. B. Postel, *User Datagram Protocol*, August 1980, RFC 768

[41] Reliable Multicast Research Group, *http://www.east.isi.edu/RMR*, 1977.

[42] S. McCanne and V. Jacobson. Vic*: A Flexible Framework for Packet Video.* In Proc. ACM Multimedia '95m November 1995.

[43] J. B. Postel and J. Reynolds. *TELNET Protocol Specification*. Information Sciences Institute, Marina del Rey, CA, May 1983. RFC-854

[44] J. B. Postel, *Internet Protocol*, RFC, Information Sciences Institute, Marina del Rey, CA, September 1981. RFC-791.

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, Jan 1997. RFC-2068.

[46] W. R. Stevens, *TCP/IP Illustrated, Volume 3*, Addison-Wesley, Reading, MA, April 1996.

[47] J. B. Postel, *Simple Mail Transfer Protocol*, Information Sciences Institute, Marina del Rey, CA, August 1982. RFC-821.

[48] J. B. Postel and J. Reynolds, *File Transfer Protocol (FTP)*, Information Sciences Institute, Marina del Rey, CA, May 1985. RFC-821

[49] W. R. Stevens. UNIX Network Programming, Addison-Wesley, Reading, MA, 1992.

[50] P. Karn and C. Partridge, *Improving Round-Trip Time Estimates in Reliable Transport Protocols*, ACM Transactions on Computer Systems, 9(4):364-373, November 1991

[51] W. R. Stevens, *TCP/IP Illustrated, Volume 1*, Addison-Wesley, Reading, MA, April 1994.

[52] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynami,* PhD thesis, U.C. Berkeley, May 1997.

[53] Mustafa A., Hassan M. and Jha S., *Design and Performance of a rate control feedback architecture*, 21st IEEE International Performance Computing, and Communications Conference (IPCCC 2002), April 3-5, Phoenix, Arizona, 2002.

[54] K. Kurata, G. Hasegawa, M. Murata, *Fairness Comparisons between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas*, to be presented at INET 2000, Yokohama, Japan, July 18-21, 2000.

[55] Lawrence S. Brakmo and Sean W. O'Malley and Larry L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*, in *Proceedings of ACM SIGCOMM'94*, pp. 24-35,October 1994.

[56] J. Mo, R. La, V. Anantharam and J. Walrand, *Analysis and comparison of TCP Reno and Vegas*, in *Proceedings of IEEE INFOCOM'99*, March 1999.

Publications

1. D. Hoang, H. Zhou, P. Nhan and V. Mirchandani, *Performance Study of a Feedback-Based Congestion Control Algorithm in Wireless Networks*, University of Technology, Sydney. To be submitted for publication.

2. P. Nhan, D. Hoang, H. Zhou and V. Merchandani, *Window-based Fair Intelligent Congestion Control with Explicit Lost Notification (ELN) Deployment*, University of Technology, Sydney. To be submitted for publication.

3. H. Zhou, D. Hoang, P. Nhan and V. Merchandani, *Feedback-Based Congestion Control and Explicit Window Adaptation in Hybrid Networks with Wired and Wireless Links*, University of Technology, Sydney. It is submitted to the IEEE Wireless Communications and Networking Conference, Atlanta, March 2003.