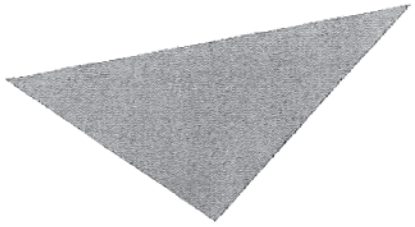
AN EVALUATION OF TCP OVER WIRED-TO-WIRELESS NETWORKS

RITESH KUMAR TAANK

Doctor of Philosophy

ASTON UNIVERSITY

October 2008



ASTON UNIVERSITY

AN EVALUATION OF TCP OVER WIRED-TO-WIRELESS NETWORKS

RITESH KUMAR TAANK
Doctor of Philosophy, 2008

Thesis Summary

Even though the Internet has evolved into a heterogeneous mesh of interconnected networks, many of the services we take for granted today, such as e-mail, web-browsing, and file downloads still depend on TCP to provide the high performance Internet services modern society has come to expect. However, this heterogeneity has become a cause for concern for standard TCP implementations in the Internet today. Today there exist the class of wireless end-users who can remain connected to the Internet without the need for a physical medium (wire). This is the *mobile and wireless networking* revolution. The field has expanded rapidly recently, particularly the uptake of IEEE 802.11 WLANs, coupled with a penetration of broadband Internet access into everyday lives of homes and offices around the world. 802.11 WLANs are typically concentrated at the edges of the Internet; hence the last-hop portion of the journey for TCP traffic arriving from the Internet must now traverse a radio (wireless) link before reaching the end-user. It is well known that wireless transmission channels are unreliable due to the general characteristics and complexities associated with the use of radiowaves as a communications link. Since TCP servers are still responsible for sending over 90% of today's Internet traffic, an understanding of the protocol's sender-side dynamics is crucial for the future dimensioning of a heterogeneous Internet. The problem is that TCP senders are unable to distinguish between a wired path and a wireless path, and are therefore unable to differentiate between losses in the wired path or losses in the wireless path, reacting to both incidents identically.

To exploit the popularity of TCP as still the dominant sender and protocol of choice for transporting data reliably across the heterogeneous Internet, this thesis explores end-to-end performance issues and behaviours of TCP senders when transferring data to wireless end-users. The theme throughout is on end-users located specifically within 802.11 WLANs at the edges of the Internet, a largely untapped area of work.

To exploit the interests of researchers wanting to study the performance of TCP accurately over heterogeneous conditions, this thesis proposes a flexible wired-to-wireless experimental testbed that better reflects conditions in the real-world.

To exploit the transparent functionalities between TCP in the wired domain and the IEEE 802.11 WLAN protocols, this thesis proposes a more accurate methodology for gauging the transmission and error characteristics of real-world 802.11 WLANs. It also aims to correlate any findings with the functionality of fixed TCP senders.

To exploit the popularity of Linux as a popular operating system for many of the Internet's data servers, this thesis studies and evaluates various sender-side TCP congestion control implementations within the recent Linux v2.6. A selection of the implementations are put under systematic testing using real-world wired-to-wireless conditions in order to screen and present a viable candidate/s for further development and usage in the modern-day heterogeneous Internet.

Overall, this thesis comprises a set of systematic evaluations of TCP senders over 802.11 WLANs, incorporating measurements in the form of simulations, emulations, and through the use of a real-world-like experimental testbed. The goal of the work is to ensure that all aspects concerned are comprehensively investigated in order to establish rules that can help to decide under which circumstances the deployment of TCP is optimal i.e. a set of paradigms for advancing the state-of-the-art in data transport across the Internet.

Keywords: **IEEE 802.11, WLAN, Congestion Control, Linux, Frame Error Rate**

To my dearest wife, Paminder…

Acknowledgements

I have always dreamt of pursuing the Ph.D. degree. Given the great effort and sacrifice that has been put into the realisation of this dream, the research presented in this thesis has only been made possible through the culmination of fantastic support I have received from the people surrounding me since the day I set out. Hence, I feel it is my duty to say thank you to all of them here, with no words spared.

First and foremost, without the constant support and encouragement from my wife, Paminder, I very much doubt if my Ph.D. thesis would have ever been anything more than just a dream. She believed in me and stood by my side, rejoicing when things went well and gently nudging me along during periods of negativity. She is my source of motivation and the force that keeps me going, daily. Therefore I would like to dedicate a very special message to my dearest Paminder; *thank you for all your patience and understanding, for the many times where I have had to work during the weekends, and thank you for always being there to support me. Ultimately, thank you for your love that has always been my main source of strength.*

Secondly, I am very grateful to my supervisor Dr. Xiaohong Peng for giving me this wonderful opportunity to pursue my research interests in the first place. Without his superior guidance, patience, and good-hearted discussions throughout my research period, it is difficult to imagine what the outcomes could have been. So thank you Xiaohong, for everything.

Being a proud member of the Adaptive Communication Networks Research Group (ACNRG) at Aston University, I have profited dearly from the many passionate discussions with my fellow colleagues over the years. With this regard, I dedicate a special thanks to Satyajit Mukherjee and Richard Haywood for their immediate availability and for always offering a helping hand upon request, which helped me to progress forward each time, maybe not always in the technical sense, but it was always of pivotal importance by reinstating the gears into motion.

The other students in my research group have also been wonderful to work alongside throughout my time at Aston University and I would therefore like to mention especially Qasim Iqbal, Tim Porter, Xingjun Zhang, and Guchun Zhang - they never hesitated to lend a patient ear or a helpful hand. Thanks guys.

I would like to express my gratitude to my parents, for providing me with a love for education and learning, and for the encouragement to pursue my interests; my brother Amar, for his never-ending faith in me and his words of wisdom; and my in-laws, for being so very understanding, encouraging and supportive at all times. I am grateful to have had all of you in my life so close to me - you all have contributed to the success of my research activity. My grandparents inspired me through their values and the courage they possessed in overcoming the daily challenges of life, and I am saddened that they did not live to see my graduation.

Finally, my acknowledgements and thanks go out to Stephen Hemminger of the Linux Foundation for his sheer patience and solid guidance in helping me to understand and master the TCP stack of the Linux kernel, as well as with the development of the various tools and techniques that made my life easier. Thanks also go out to CACE Technologies for providing me with equipment at subsidised rates, which proved very useful.

Table of Contents

List of Figures and Tables

Chapter 1

Introduction

The Internet has evolved into a global phenomenon allowing users across the world to share information in a fast, convenient, and reliable manner. It was the transformation of a small collaborative computer network belonging to researchers at the US *defense advanced research projects agency* (DARPA) into a distributed packet-switching communication system that offered fast end-to-end transmissions of data between end-hosts, which set the foundations for the modern-day Internet and accompanying protocols.

Deep within these foundations were the beginnings of a protocol that would dominate and govern the end-to-end transportation of Internet traffic in a reliable manner for many years to come; the *transmission control protocol* (TCP) was formally born in 1981 [1]. Today the Internet is still an evolving network, and many of the services that we now take for granted, such as e-mail, web-browsing, and file downloads/transfers, still depend on TCP to provide the pervasive Internet services modern society has come to expect [2]. It is TCP's pioneering congestion control mechanisms that have heavily contributed to and still govern the movement of over 90% of data across the Internet today [3] [4] [5] [6].

As the Internet has grown, there were developments of application level protocols to support the ways in which information was handled and presented to end-users. It is with no surprise then that the majority of implementations of such protocols have been based on TCP and still depend on it to provide a guaranteed and reliable transmission service between sending hosts and receiving hosts over the Internet [7] [8], which is very much still the case today.

Today's global Internet has the major characteristics of one that continues to grow rapidly, evolving into a huge mesh of interconnected sub-networks, and one that is increasingly becoming heterogeneous in a way that was never predicted in the

Internet's early days [9]. It is this heterogeneity that has become a cause for concern for the legacy TCP [10]. When referring to the term heterogeneous we are referring to a characteristic that is difficult to describe with absolute precision. However, certain things are clear; traditionally, it was always certain that the transmission paths (links) between communicating hosts were of the wired form. Nowadays, however, the links which make up the Internet are an intertwining of copper wires to state-of-the-art optical fibres, from infrared-based to radio-based (wireless) links, each with its own transmission properties and path-loss characteristics [11] [12] [13]. As a direct result, today there exists a new class of connected end-hosts; the class of wireless end-users who can remain connected to the Internet without the need for a physical medium (wire) between them and their nearest gateway. This is the *mobile and wireless networking* revolution [14] [15].

The mobile and wireless networking field has attracted a great deal of interest in recent years [16] [17] [18] [19]. As a result, the field has expanded rapidly in the last decade, particularly with the recent introduction and rapid uptake of the IEEE 802.11 standards for indoor *wireless local area networks* (802.11 WLANs) [20], coupled with a booming penetration of broadband Internet access into everyday lives of homes and businesses around the world [21]. IEEE 802.11 WLANs are typically found at the edges of the Internet within the homes and offices of end-users; the technology allows end-users to remain connected to the Internet via a local gateway, as in a traditional LAN, but without the cables, i.e. the radio medium is used [22] [23]. Hence, the last-hop portion of the journey for TCP data traffic arriving from a server in the Internet often now traverses a wireless link before it reaches the end-user, i.e. TCP receivers can now be wirelessly connected [24]. However, it is a well known fact that wireless transmission channels are notoriously unreliable for the transmission of data due to the general characteristics and complexities associated with the use of the radio medium as a communications link [25]. It has been shown and proven many times over the years that there are diminishing effects of wireless channels on TCP performance, causing it to under-perform due to the highly variable and unpredictable conditions which they exhibit [11] [13] [26] [27] [28] [29] [30] [31]. For these reasons, the interaction between TCP and its usage over IEEE 802.11 WLANs specifically has been the focus of much research in recent years [32] [33] [34] [35] [36] [37] [38] [39] [40] [41]. Unfortunately, the majority of such studies do not properly address the impacts of this

changing wireless last-hop scenario on TCP's performance and its consequent behaviour at the sending side in the Internet (i.e. at the servers), which is mainly due to the difficulties associated with gaining access to server machines in the Internet [42] [43]. Figure 1.1 illustrates the changing end-to-end scenario for TCP connections over wired-to-wireless paths.

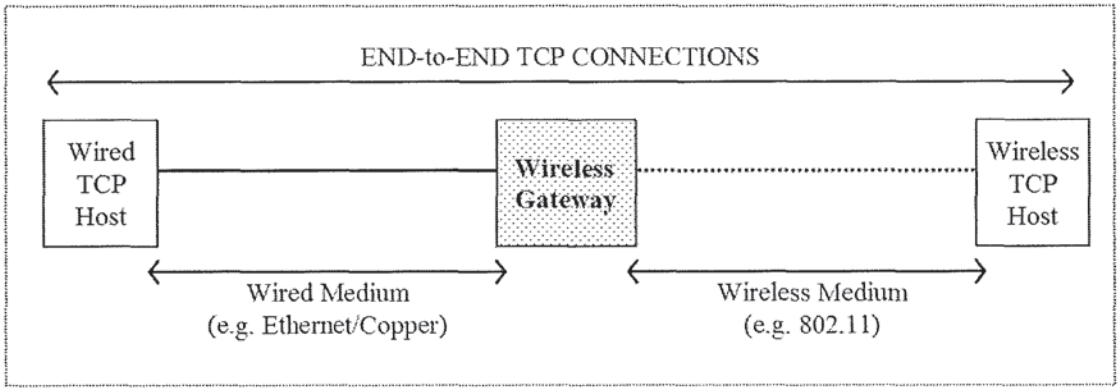


Figure 1.1: The changing end-to-end scenario for TCP connections in the heterogeneous Internet

Since TCP still stands as the dominant transport protocol responsible for over 90% of today's Internet traffic, an understanding of the protocol's sender-side dynamics is crucial for the future dimensioning of a heterogeneous Internet [2] [44]. To briefly clarify, sender-side TCP implementations typically maintain the end-to-end semantics stated in TCP's original design [45] [46], an important feature which has helped it to work seamlessly with other protocols for years. It basically implies that TCP must operate transparently between communicating hosts without any knowledge of the transmission path being used to transfer the data. Due to this lack of discrimination by TCP, it is unable to distinguish between a wired path and a wireless path, and is therefore unable to differentiate between data losses in the wired path and data losses in the wireless path. It reacts to both incidents in an identical manner. This matters because TCP has been designed to deal with wired losses by throttling back its sending rate as it assumes congestion has occurred. For wireless losses, this response from TCP would unnecessarily reduce its sending rate as there is no congestion in the path, and wireless losses can normally be recovered locally using a variety of error recovery techniques.

In summary then, this thesis explores the end-to-end performance issues and behaviour of sender-side TCP implementations when it is transmitting data towards a requesting wireless end-user. The focus throughout is on those end-users that are located specifically within IEEE 802.11 WLANs, where the last-hop path for all TCP data traffic must traverse a radio link at the edges of the Internet, a relatively untapped area of work. The thesis also makes an attempt to explore the transmission characteristics of IEEE 802.11 WLANs, with the hope of gaining a better understanding of its interactions with TCP's sending behaviour.

1.1 The Challenges and Motivations

1.1.1 Evolution of 802.11 WLANs and the Impacts on TCP Senders

The end-user experience for accessing services from the Internet is changing; WLANs based on the IEEE 802.11 standard have seen an explosive growth in recent years. WLANs have enabled end-users to become mobile whilst remaining connected to a wired backbone, without the need for any wires. The license-free usage of radio spectra in the 2.4 GHz ISM band has aggressively encouraged WLAN technologies to be adopted into homes and offices around the world [32] [47]. More and more manufacturers are incorporating 802.11 WLAN capabilities into an ever-increasing array of electronic consumer devices. As a result, the Internet has become increasingly heterogeneous, and is set to continue with this trend for the foreseeable future [10].

Typically, a WLAN is situated at the last-hop portion of the end-to-end journey for data arriving from Internet servers [47]. It implies that a communication path for the transfer of data from the Internet to an end-user may traverse a wired (or fixed) path for majority of the journey, and then a wireless path in the final stage of the journey. Researchers often refer to such scenarios as 'wired-cum-wireless' environments [27].

It is well known that indoor 802.11 WLAN transmission channels exhibit higher bit error rates (BERs) due to the degradation of radio signals from fading, attenuation, reflections, refractions, diffractions, and the addition of interference noise [25]. Hence the performance of a WLAN is highly dependent on the characteristics of the radio channels which it uses. To counter these effects and provide a reliable service the IEEE

802.11 standard implements its own error detection and recovery mechanisms at the 802.11 MAC-layer (MAC) over the WLAN [23].

It is clear that Internet access technologies have become diverse; however, certain factors remain unchanged. End-users still need and expect fast and reliable downloads of data from servers in the Internet, and TCP still dominates on servers as the transport protocol of choice for supporting these activities [48]. Unfortunately, TCP was primarily designed with only the fixed Internet in mind, and has been fine-tuned over the years to deal with congestion-related losses and the very low BERs typical of wired paths via its congestion control algorithms [49].

The key issue then for TCP sender over last-hop 802.11 WLANs is the higher random loss rates that TCP traffic is subjected to whilst in transit over air. Simply put, a TCP sender in the Internet is unable to discriminate between a loss in the fixed network, and a loss occurring in the WLAN [13]. It detects and reacts to both types of losses in the same manner, activating its congestion control procedures to alleviate the assumed congestion in the fixed network, by bringing down its sending rate [11]. Of course, this is the correct action if the loss was indeed over the wired path, as congestion may be imminent. However, and more likely than not, if the loss was over the WLAN then a reduced sending rate by TCP is inappropriate because 802.11 WLANs typically continue to send data to its wireless clients at the same data rate. All that is required of TCP in this situation would be to simply retransmit the lost segment. In summary then, constant losses over the WLAN caused by high BERs can lead to sub-optimal average data throughputs for end-users due to a TCP sender's inherent inefficiencies.

Another issue for TCP senders over last-hop 802.11 WLANs is the highly variable delays experienced by TCP segments due to a) the 802.11 error recovery mechanisms in place, and b) the *medium access control* (MAC) mechanism governing client access to the radio channel for transmissions. Firstly, as a direct consequence of the aforementioned higher error rates, the 802.11 MAC utilises a persistent retransmission technique to deliver lost data locally, which can cause unexpected and considerable transmission delays for queued data [50] [51] [52]. Secondly, when there are many wireless 802.11 devices in the WLAN then access to the radio medium for transmissions becomes an issue; the 802.11 MAC allows only one device at a time to

transmit/receive data at any one time. When there are multiple 802.11 devices in the WLAN, gaining access to the radio medium is controlled by an exponentially increasing random waiting period [53]. Consequently, depending on which device in the WLAN a fixed TCP sender is communicating with, there could be unpredictable waiting times between transmissions of individual packets of data from the Internet.

It can be argued that each of these factors can have a direct impact on the end-to-end delays experienced by a TCP sender. In the traditional fixed Internet of the 1980's, end-to-end *round trip times* (RTTs) for TCP segments remained fairly constant with only small deviations from the mean [3]. This relatively stable behaviour allowed TCP to always remain in a steady state, constantly adjusting its sending rate and its loss detection sensitivity based on measured RTTs, where the imminence of network congestion is always on TCP's agenda. Now, however, when TCP experiences highly variable RTTs associated with WLANs, it loses its steady-state behaviour because its congestion control mechanisms are insensitive to any other form of delay than congestion related delays [31]. As a result, TCP senders are often forced to make unnecessary retransmissions due to mixed signals about delay conditions in the network, thereby reducing its average sending rate over the course of a connection.

To date, many reports have been published highlighting the performance implications of using the legacy TCP over wireless paths [11] [13] [26] [27] [28] [29] [30] [31]. However, the majority of these studies are limited because they do not look at the IEEE 802.11 wireless technology in particular, which is a core focus of the work in this thesis [54] [55] [56] [57]. Those that do study its behaviour over IEEE 802.11 WLANs provide only basic insights into the performance of TCP senders, and focussing mainly on TCP versions and settings that are deprecated, and not providing enough insights into the sending behaviour of TCP servers separated by an Internet (or by a large wired backbone). Such studies also tend to overlook the characteristics of the 802.11 WLAN that affect TCP's performance in particular, skipping discussions relating to the interactions between the two protocols. Therefore, due in no small part to the fact that downloads by end-users (both wired and wireless) of TCP application data account for the bulk of Internet traffic there is a paramount need for an up-to-date study and performance evaluations of the behaviour of modern-day TCP senders from

the perspective of Internet servers, when the last-hop portion of connections includes IEEE 802.11 WLANs.

1.1.2 Testing Wired-to-Wireless TCP in More Realistic Conditions

An area that has gathered more interest in recent years is that of performing real-world and hence more accurate experiments with networking protocols as a supplement or as an alternative to traditional methods of performing experiments. Currently the most popular method of performing TCP experiments over wired-to-wireless paths is by using network simulation software [58], followed by using the concept of network emulation techniques [59].

Naturally, simulation has always been the platform of choice for conducting such experiments, with popular simulators such as NS-2 [60] and OPNET Modeler™ [61] dominating in the arena. These simulator suites offer excellent features and provide accurate implementations and models of both TCP and the IEEE 802.11 WLAN protocols. Such features make simulation an easy choice for new researchers wanting to study and experiment with protocols in a controlled manner, allowing them to rapidly create network topologies, generate customised traffic patterns, and seamlessly collect and analyse data. Simulators also have the advantages of being relatively cheap, providing highly reproducible results, and scaling very well and inexpensively [10]. However, simulators can be artificial and offer synthetic environments only [10] [62] [63], although it is appreciated that these traits could be viewed upon as advantageous.

Network emulation is a technique for conducting experiments using real machines and devices; the components are connected to form a system that basically mimics the behaviour of an equivalent system in the real-world [58] [64] [65] [66]. For example, a machine could be configured to emulate the effects of a wireless network by dropping packets based on customised settings that seem appropriate for a radio channel. Emulators are advantageous because researchers can observe real-world protocols in action; they allow controlled experiments with a high degree of reproducibility. Generally speaking, emulators sit between simulators and real-world live systems. On the downside, emulators tend to be quite specialised systems, built by researchers for a

particular study at hand [67] [68]. Hence, emulators generally are not easily transferrable between researchers.

In light of many of the advantages of network simulation and emulation, modelling the true behaviour of TCP running over wireless networks should be a precise procedure, and one that simply cannot be carried out using simulation models or emulation alone. From the vantage point of TCP research over wired-to-wireless paths, the greatest challenge has always been the ability to experiment with real-world protocol implementations using a real-world operational wireless network and over conditions that most accurately reflect the real-world, i.e. the Internet. The challenge is in the acquisition of equipment/devices that are readily available to researchers, as well as having support for the usage of easily obtainable software measurement tools.

A further challenge is being able to observe and measure the behaviour of a sending TCP whilst downloading data from a server in the Internet. As can be appreciated, this is a complex task in the real-world because of the vastness of the Internet, as well as not knowing precise physical locations of the servers. Coupled with this challenge, it would be very beneficial if protocols used over the wireless portion whilst making the download requests could be observed/studied simultaneously to TCP senders. Such a setup would allow researchers to cross-compare performance between the two protocols for the same transfer (or experiment), investigating any relationships on the interactions between the two protocols. A thorough literature review of previous experimental work on TCP over wired-to-wireless paths suggests that an accurate testing technique for evaluating both wired and wireless protocols in unison does not readily exist.

The challenges presented above would require the use of real-world protocols and live networks in order to supersede simulation or emulation techniques. However, if possible, such an experimental platform would provide researchers with innovative ways of experimenting with TCP over wired-to-wireless paths, which is important for the future of TCP's robustness in the heterogeneous Internet.

1.1.3 Transmission Characteristics of Indoor IEEE 802.11 WLANs

The adoption of IEEE 802.11 WLANs into homes and offices has led to a changing scenario for higher layer protocols, specifically for TCP. Such in-building WLANs in this context utilise indoor radiowave channels that are available in the 2.4 GHz ISM band [47].

Indoor radio channels differ from traditional outdoor radio channels in two key aspects [25]; i) the distances travelled by radiowaves are much smaller, and ii) the variability of the radio environment is much greater for a smaller range of transmitter-receiver distances. Although indoor radiowave propagation is subjected to many of the same mechanisms as outdoor channels, such as reflection, diffraction, and scattering, the added variability factor is that radiowave propagation within buildings is strongly influenced by specific factors such as the layout of internal walls/partitions, the construction materials used, and the overall layout of the building. For example, indoor wireless signal levels can vary greatly depending on whether interior doors are open or closed. In fact, virtually every component within a building can have an affect on propagating radiowaves, by either reducing their propagating transmission power (known as path loss) or by interfering with them and altering transmission properties (known as interference noise).

Another feature that is unique to homes and offices today is that they are often loaded with electronic devices that act as additional sources of noise interference to radiowave transmissions belonging to an 802.11 WLAN [69] [70]. In particular, the license-free nature of the 2.4 GHz spectrum has led to the development of many consumer devices that can also take advantage of wireless communications [47]. Typical devices operating in the 2.4 GHz frequency band that can be found in the majority of homes and offices today include digital cordless telephones, wireless printers, wireless audio-visual equipment, and microwave ovens [71]. It should also be mentioned that due to the high popularity and penetration of IEEE 802.11 WLANS, they too can act as sources of interference. For example, in a large multi-floor office building there may be several WLANs installed throughout the building. Due to restrictions in certain countries on the maximum number of channels that can be used in the 2.4 GHz band by IEEE 802.11 equipment, it is easy to understand why an overlap on the same

wireless channel between two neighbouring WLANs is easily possible [72]. This situation also applies to residential sites that are heavily built-up with many houses clustered within small areas. For example, an 802.11 WLAN operating in one house can easily interfere with the radiowaves of an 802.11 WLAN in a house next door.

The unique characteristics of indoor radio channels mentioned above lead to very specific behaviour with the transmission of data over 802.11 WLANs. The ultimate measure of the impacts of channel characteristics is the error rate experienced by data frames over the WLAN, known as the 802.11 *frame error rate* (FER). It is well known that wireless channels generally possess higher error rates than fixed networks [16], but there has always been interest from researchers wanting to gain more accurate and comprehensive insights into the characteristics and FERs of real-world indoor 802.11 WLANs specifically. Such insights can be very useful to researchers, for example when developing and enhancing existing wireless channel models that are commonly implemented in network simulators or emulators.

To illustrate, when TCP traffic traverses over an 802.11 WLAN, each data unit is encapsulated within at least one 802.11 frame before being transmitted over the air. Hence, any errors occurring with an 802.11 frame will always affect the TCP data contained within it, potentially leading to a presumed data loss by the TCP sender.

Several studies have been undertaken to understand a) the transmission characteristics of frames over 802.11 WLANs [73] [74] [75] [76], and b) their subsequent impacts on higher-layer protocols [33] [34] [77] [78] [79] [80]. However, in the case of a), most of the studies have not been comprehensive enough, and therefore have only produced very basic insights regarding typical conditions of indoor WLAN channels; the common trends by researchers are to focus only on one-way traffic directions over the WLAN, either the *downlink* (towards the end-devices) or *uplink* (away from end-devices) traffic flows. Assumptions are also made about the channel conditions being used, with some researchers still using older 802.11 standards that have been superseded in the marketplace, as well as using basic techniques for the analysis of captured data leading to inaccurate insights. In the case of b), there appears to be a lack of parallelism with the real-world with how higher-layer protocols such as TCP are actually used over WLANs. The majority of studies in this area also tend to use

network simulation studies to evaluate and highlight the impacts of 802.11 WLANs on higher-layer protocols, with some resorting to the formation of mathematical models to predict higher-layer protocol performance. Thus, insights into the error and transmission characteristics of indoor IEEE 802.11 WLANs remain of considerable research interest, as well as interest into the techniques used.

1.1.4 Screening Linux TCP Implementations for Wired-to-Wireless Paths

Linux has evolved into a powerful operating system, and its popularity as a license-free and open-source operating system has gathered momentum in recent years. The current major release of Linux is *version 2.6*, and it comes armed with many potent features aimed at improving the performance of enterprise web-servers. One of these features is the leading edge implementation of a powerful networking stack, in particular its support for a flexible TCP [81]. Secondly, because the Linux framework is customisable it has encouraged researchers of networking protocols to take advantage of this flexibility by contributing different implementations of TCP's congestion control algorithms over the years [82] [83]. Hence, recent sub-releases of the Linux v2.6 kernel now come with an array of ready-to-use TCP congestion control variants, which can be activated with relative ease at runtime. Some of these implementations are aimed at achieving high-speeds and bandwidths across the Internet, whilst others have been developed for use over wireless paths [84].

The main motivations for studying TCP in Linux v2.6 arise from the fact that Linux offers a very robust and a freely available native web-server, by the name of *Apache HTTP* [85]. Apache is the most widely used web-server in the world today, and has been since the year 2000 according to *Netcraft Limited*, an Internet research agency. Netcraft's research on the total number of active web-servers seen across all Internet domain names globally suggests that in terms of popularity Apache surpasses all other competitors in the market. For this reason, it could be said that end-users accessing and downloading content from the Internet are more likely to be served by TCP on web-servers running Linux. In today's heterogeneous Internet, increasingly these end-users will also be those connected to the Internet wirelessly via last-hop 802.11 WLANs.

Thus, there is a real need to study and constructively evaluate how the various TCP sender-side implementations in Linux v2.6 perform in the real-world when a Linux server is subjected to TCP connections that include a wireless link in the last-hop portion of the path. Currently there are only a handful of previous studies investigating Linux TCP implementations in the real-world [84] [86]; however many of these studies are focussed only on performance over fixed/wired networks. In contrast, studies using the recent v2.6 Linux kernel involving a last-hop 802.11 WLAN could not be easily found. Such insights can help researchers to re-evaluate their TCP implementations, encouraging the Linux community to hold onto the fittest implementations, and eliminate those under-performers from future releases of the Linux kernel. In the long run, this can only lead to a better quality of service for wireless users of the Internet.

1.2 Objectives of the Thesis

The core objectives and focus of the thesis are:

1. To exploit the general popularity of TCP, which still remains as the dominant transporter of application data across today's heterogeneous Internet. This leads onto a need for investigating the sending behaviour of fixed TCP senders when the end-users it serves are positioned in 802.11 WLANs at the last-hop portion of end-to-end connections, constantly demanding increasing amounts of data with expectations of performance that resemble a typical wired client.

2. To exploit the recent interest amongst researchers wanting to study the true behaviour and performance of TCP over such heterogeneous conditions; this thesis aims to highlight and propose a platform and a systematic methodology for more accurate testing with TCP under conditions that better reflect the real-world.

3. To exploit the transparent functionalities between TCP senders in the wired domain (i.e. on a server somewhere in the Internet) and the IEEE 802.11 MAC protocol, usually operating in WLANs locally within homes or office environments; this thesis aims to explore the transmission and error characteristics of real-world indoor 802.11 WLANs, and aims to correlate any findings with the functionality of TCP.

4. To exploit the popularity of Linux as a popular operating system for many of the world's Internet servers; this thesis aims to study the implementations of TCP sender-side congestion control variants within the most recent Linux kernel release (v2.6). It aims to look at the various TCP enhanced implementations (aimed at both wired and wireless paths) that have made their way into an increasingly popular operating system. A selection of these real-world TCP implementations will be put under test in real-world wired-to-wireless conditions in order to screen and present a viable candidate for further development and use in the modern-day heterogeneous Internet.

1.3 Thesis Structure

This thesis presents a comprehensive study and experimental investigations into the behaviour and performance of TCP when it is sending data to requesting wireless end-users located in last-hop IEEE 802.11 WLANs. The focus throughout the thesis is on the download action of TCP application data, and the resulting impact on the TCP sender due to the radio link in the end-to-end journey of TCP traffic. A common theme throughout is the idea of TCP connections traversing a 'wired-to-wireless' path, a modern concept that builds on the traditional ideology of TCP connections being completely 'wired'.

Chapter 2 gives a brief introduction to the TCP protocol, starting from the very fundamentals, and building up to its essential features that will form the focus in the rest of the thesis. Due to the fact that this thesis spans across two very different networking fields, Chapter 2 also presents an overview of the IEEE 802.11 WLAN technology; again only brief insights into its key features and functionalities relevant to the thesis are given.

Chapter 3 introduces the problems associated with TCP and its usage over wired-to-wireless environments, forming a strong rationale for continued research into the performance issues associated with its non-standard behaviour. The specific characteristics of IEEE 802.11 WLANs that make things challenging for TCP are reviewed and succinctly stated. The various approaches taken by researchers to

enhance the performance of the legacy TCP over wired-to-wireless paths are reviewed. Finally, an up-to-date literature survey of the studies on the performance of TCP specifically over IEEE 802.11 WLANs is provided.

Chapter 4 introduces the increasingly popular OPNET Modeler™, an alternative network simulation suite, with its own implementations of TCP and the IEEE 802.11 protocols. The aim of this chapter is to confirm TCP's performance suffering in wired-to-wireless paths, but more importantly to probe how resistant TCP is to radio channel bit errors over a last-hop 802.11 WLAN. A custom wireless channel error model is developed, replacing the simulator's standard error model, which is then used to test TCP's resilience under varying error rates. Simulation results using a typical WLAN topology are presented with discussions highlighting the impact of channel conditions on the performance of a TCP sender, as well as the impact on popular end-user applications which utilise TCP.

Chapter 5 builds on the conclusions from the previous chapter by investigating the impacts on TCP sending performance by subjecting end-to-end connections to bidirectional path errors, as opposed to unidirectional errors which many researchers tend to use in their experiments. Due to the fact that wireless channels affect TCP traffic flowing in both directions, it was necessary to highlight the significance of excluding a feedback channel loss model for TCP experiments. An end-to-end TCP emulation platform is used to conduct comprehensive experiments to fully investigate and confirm the theory. Experimental results of TCP sender performance is presented, followed by discussions of comparisons between unidirectional and bidirectional loss models.

Chapter 6 relates to the exploitation of researchers increasingly wanting to study and experiment with TCP in realistic conditions over a wired-to-wireless environment. Initially, a review of the most popular techniques currently being used by researchers is presented, ending with a clear rationale for the need for more advanced experimental platforms. A wired-to-wireless experimental testbed for TCP studies is consequently proposed, with full descriptions of its key components and features. Further specific knowledge in the area of IEEE 802.11 WLAN functionality is then presented, building on the information given in Chapter 2 and Chapter 3. Using the proposed testbed, a

further focus in Chapter 6 is on the transmission and error characteristics of 802.11 frames in real-world WLAN conditions, and how they correlate with the behaviour of a TCP sender that could be located in a geographically different location. A unique methodology for calculating downlink and uplink channel frame error rates over the WLAN is proposed. Further to this, a technique for extracting the distribution of retransmission probabilities for a transmitting 802.11 MAC is also proposed. Such information is useful because it allows researchers of higher-layer protocols, such as TCP, to get a better understanding of conditions that TCP needs to withstand. A wide range of results and analyses of many experimental runs are presented, highlighting the capabilities of the proposed testbed. A comparison of frame error rates calculations against theoretical transmission delay values is presented, as well as insights into the retransmission behaviour of TCP in correlation with the frame error rates.

Chapter 7 presents a comprehensive evaluation of real-world TCP congestion control implementations that have made their way into the most recent Linux v2.6 operating system. Due to the popularity of Linux on web-servers across the Internet, it seemed logical to conduct a study into the various TCP implementations it currently hosts. TCP Reno and TCP CUBIC were chosen as they are the most popular algorithms for wired data transfers, so were TCP Hybla, TCP Veno, and TCP Westwood+ because of their respective authors claim of enhanced performance over wired-to-wireless paths. The setup procedures for the experimental testbed consisting of a real-world Linux server and a last-hop IEEE 802.11 WLAN are described, alongside preliminary results from experiments evaluating error conditions in the WLAN prior to running the TCP experiments. Finally, two real-world testing scenarios are chosen that best reflect a modern-day home-office WLAN; i) multiple end-users in the WLAN, and ii) varying the location within the home-office of a WLAN end-user. The performance results, including detailed evaluations of how the TCP implementations perform in each of the scenarios alongside discussions then follow.

Chapter 8 finally concludes the studies of TCP sender behaviour over wired-to-wireless paths, clarifying the key findings and contributions relating to the four key exploitations that form the focus of the thesis. Some recommendations for future work resulting from the works in the thesis are then provided, encouraging researchers to

take their studies and understanding of TCP over heterogeneous environments, and hence its performance, to the next stage.

Chapter 2

Background Information

The objective of this thesis is to investigate and evaluate the behaviour and robustness of TCP when wired connections originating from a server in the Internet traverse a wireless link in the last-hop path, specifically over an IEEE 802.11 WLAN. Figure 2.1 illustrates the typical scenario TCP servers in the Internet have to deal with when sending data to end-users located in a WLAN at the last-hop.

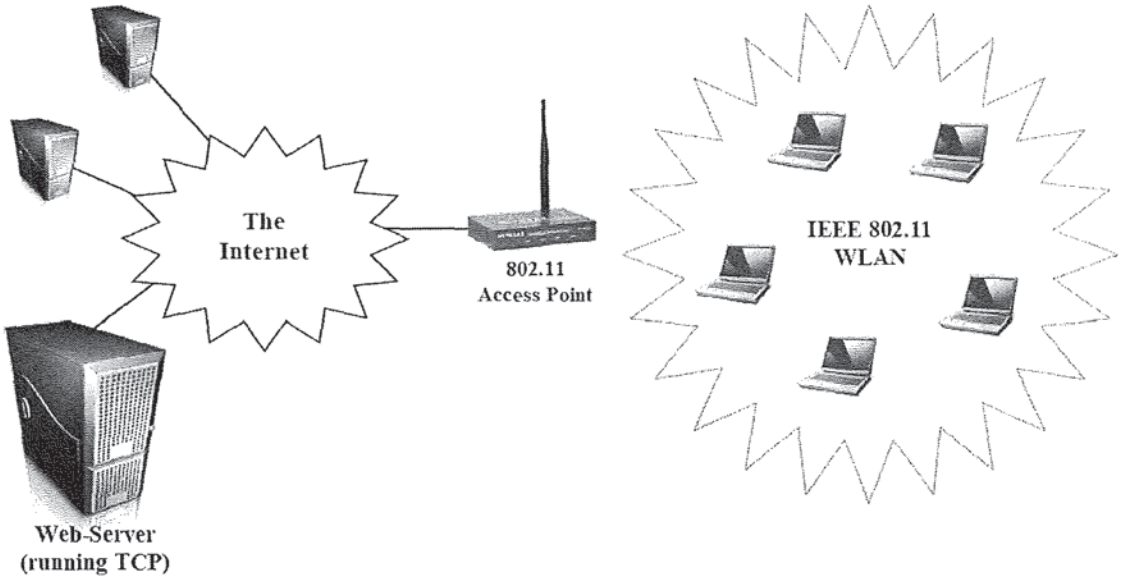


Figure 2.1: A typical last-hop WLAN path scenario for TCP servers in the Internet

To fully understand the work and terminologies used throughout the thesis, one must understand several important concepts that make up TCP and IEEE 802.11 WLANs. Due to the fact that this thesis spans across two traditional fields, this chapter will provide a brief review of the building blocks and mechanisms of TCP, as well as an overview of the functionality of IEEE 802.11 WLANs.

This chapter is organised as follows: an introduction to the significance of TCP is presented, alongside details of its evolution and insights into the importance of *congestion control* in the Internet. Next, the workings of TCP's *AIMD* congestion

control algorithm are broken down and summarised, ending with a brief look at the state of TCP in the Internet today. Finally, the chapter presents an overview of the IEEE 802.11 standard, concentrating specifically on *infrastructure-mode* WLANs. An introduction to the functionality of the 802.11 MAC is then followed by brief insights into the 802.11 PHY-layer (PHY) internals. The section ends with some details and terminologies associated with 802.11 WLAN radio channels.

2.1 TCP – An Overview

Transmission Control Protocol (TCP) remains as the dominant *end-to-end* transport protocol used in the Internet today [46], and its most basic working principles have not changed significantly since the draft of its original specification in 1974 [87] and its official birth in 1981 [1].

In essence, TCP provides a reliable, full-duplex in-sequence delivery of data [88]. Its popularity arises from the fact that from the beginning it has been used by an array of applications requiring a reliable data transmission service between communicating end-hosts, including e-mail (SMTP), file transfers (FTP), virtual terminal service (Telnet), and for web-page transfers from the Internet (HTTP).

TCP's reliability comes from the fact that for every *segment* of information that is sent, the sending host expects to receive an *acknowledgement* segment from the receiving host upon successful reception of the information [49].

Without a transport layer such as TCP the Internet would be based on a connectionless communication model of the Internet Protocol (IP) [89]. The IP protocol does not itself provide mechanisms to guarantee the delivery of data, and hence through the encapsulation of TCP segments it is able to transfer packets across a network reliably via TCP's end-to-end error detection and recovery mechanisms [49].

2.1.1 Evolution of TCP and Internet Congestion Control

Since the dawn of the Internet, TCP has been phenomenally successful. Today it controls the transmission of most traffic on the Internet [3] [4], everything from the downloading of web pages to the transfers of bulk data, and even peer-to-peer file sharing applications.

As already stated, TCP's origins go back several decades, but it wasn't until the *congestion collapse* of the Internet in the mid-1980's [90] that TCP first went through a major revision. It was between the years 1987 and 1988 when TCP end-hosts began to adopt new algorithms in order to cope with the pressures of an Internet reaching over-capacity caused by fast TCP senders and unnecessary retransmissions. The new mechanisms were known as the *congestion control and avoidance* algorithms and were proposed by Jacobson [91]. The key algorithms in concern were labelled with the titles of *slow start*, *congestion avoidance*, and *fast retransmit*, and the versions of TCP implementations which adopted them were named TCP Tahoe. This major event for TCP led to many performance studies being undertaken by a curious Internet community on the congestion control algorithms [88] [92], and shortly afterwards in 1990, Jacobson suggested further modifications to TCP implementations with the addition of the *fast recovery* algorithm [93]. From here onwards, versions of TCP which also adopted the fast recovery algorithm were named *TCP Reno*.

Throughout the 1990s, performance studies on TCP and the behaviour of its congestion control algorithms in the Internet continued [94] [95] [96]. Eventually, in 1997, a *request for comments* (RFC) document was submitted to the Internet community by Stevens in an attempt to bring some form of cohesion amongst TCP implementations in the Internet [97]. The aim of the RFC was to clear up confusions relating to the workings and requirements of the legacy congestion control algorithms. Based on a comprehensive study of TCP issues [98], Steven's RFC document was later made obsolete by an updated and more concise RFC document in 1999 [99], clearly describing the workings of Jacobson's original TCP algorithms and their significance in maintaining a stable Internet. The document became one of the most important TCP related RFCs in recent years, and is still adhered to in the modern-day Internet by developers of new TCP features.

Today, TCP Tahoe and TCP Reno are referred to as the *legacy* TCP versions, with TCP Reno still being widely used in legacy systems due to its popular uptake throughout the 1990s [8] [100]. Arguably, Jacobson's congestion control algorithms have been the reasons for making the Internet succeed, by coming to its rescue. However, there is sentiment amongst the Internet community that Jacobson's algorithms are beginning to show their age [48], and that perhaps it is time for another major upgrade if TCP is to continue to support a future Internet which is becoming increasingly heterogeneous.

To understand why the legacy TCP has been so successful, and to justify why an upgrade may be necessary, it might be useful to know that today TCP code is embedded into every machine and device connected to the Internet. TCP has been implemented into dominant operating systems including *MS Windows* and Linux, as well as into mobile phones and portable devices. So everybody sending or receiving data to or from the Internet today, in some way or another, is dependent on TCP to provide them with a quick and reliable service.

2.1.2 Key Features of TCP

In this sub-section the key operating features of the legacy TCP Reno as a reliable transport protocol are briefly summarised for the reader.

2.1.2.1 End-to-End Semantics

One of the reasons for the success and growth of the Internet is due to the *end-to-end* aspects of TCP's congestion control mechanisms [45] [101]. This rule (or argument) states that any complexities should be moved 'out of the network' where possible, i.e. towards end-points, and as high as possible in network protocol stacks [3]. Thus, with TCP, it implies that all communicating end-hosts of a network are allowed to send data to any other host without requiring intermediate knowledge of network conditions or of its elements [102].

2.1.2.2 Ordered Data Transfers

TCP maintains an ordering system when it transmits data *segments* by using a system of incremental *sequence numbers* to identify each byte of data that is sent [1]. This helps TCP keep track of the order in which data was sent so that it can provide a reliable and in-order transmission service, regardless of any problems that may occur along the communication path. TCP is therefore able to deal with issues such as disordering, fragmentations, and complete losses of segments.

2.1.2.3 Cumulative Acknowledgements

TCP uses a *cumulative acknowledgement* scheme, where the receiver must always send an acknowledgement (ACK) segment indicating that it has received all data bytes preceding the sequence number it is acknowledging [49]. Essentially, the first data byte in a segment is assigned a sequence number, which is inserted in the sequence number field of a segment's header. The receiver then sends an ACK segment specifying the sequence number of the data byte it expects to receive next, inherently indicating all bytes up to this sequence number have been successfully received.

2.1.2.4 Round Trip Timing

A TCP sender works towards providing a reliable data delivery service by measuring the *round trip time* (RTT) of sent data segments. The RTT of a data segment is the time interval between sending the segment and receiving an acknowledgement for it [49]. Experience has shown that accurate and current RTT estimates are necessary for a TCP sender to adapt to changing traffic conditions and, without them, a busy network is prone to instability [103]. The RTT is therefore considered a deterministic characteristic of the communication path in the underlying network.

2.1.2.5 End-to-End Flow Control

TCP uses an end-to-end window-based flow control mechanism to avoid situations where fast sender machines transmit too much data too quickly to slower TCP receivers, allowing receivers to reliably receive and process the incoming data [99]. A window-based protocol implies that the current size of a window defines a strict upper-

bound on the amount of unacknowledged data that can be in transit between a pair of communicating TCP hosts [49]. Having such a mechanism is essential for TCP in environments where fast sending machines dominate in the sending of data towards end-hosts, such as in the Internet.

As documented in [99], the congestive collapse of the Internet gave rise to two important sender-side variables to be used in conjunction with the congestion control algorithms; i) the *congestion window* (*cwnd*), and ii) the *slow start threshold* (*ssthresh*). The *cwnd* at a TCP sender is adaptively increased or decreased based on the sender's view of current network load conditions in the communication path [49]. These variables implement a sending window, W, whose size is defined by:

$$W = \text{minimum}(cwnd, rwnd) \qquad \textbf{(Eq. 1.1)}$$

where *rwnd* is the receiver's advertised window size. In essence, the size of W prevents a sender from injecting more data into a communication path than the network or receiver can accommodate. The *ssthresh* variable will be discussed in the next section.

2.1.2.6 AIMD Congestion Control

A final key feature of TCP is its congestion control algorithms, whose purpose ultimately is to achieve high performance in the transfer of data across the Internet by keeping network path conditions in a state of equilibrium [49]. As already stated, modern implementations of TCP should contain four intertwined congestion control algorithms [99]: i) *slow start*, ii) *congestion avoidance*, iii) *fast retransmit*, and iv) *fast recovery*. When the four algorithms are combined and used in union they are frequently referred to as the *additive-increase multiplicative-decrease* (AIMD) algorithm. The AIMD algorithm represents a linear growth of a TCP sender's *cwnd*, combined with its exponential reduction when segment losses due to congestion events occur. A segment loss event is generally described to be either a *retransmission timeout* (RTO) or the event of receiving three *duplicate-ACKs* (DUPACKs) [49]. The RTO event and DUPACKs will be discussed in the next section.

2.2 TCP Congestion Control in the Internet Today

In this section the workings of the congestion control algorithms used by the popular TCP Reno in today's Internet, as defined in [99], will be briefly described.

2.2.1 Slow Start

Slow start was designed as a fix for the original TCP's [1] aggressive start-up behaviour, in which fast senders were quickly flooding the communication path to over-capacity, leading to segment losses due to buffer overflows at overwhelmed intermediate routers and very poor throughput performance for TCP applications [3].

In short, the algorithm allows a TCP sender to quickly, yet reassuringly increase the rate of injection of data into the network [49]. It allows a connection to probe for free bandwidth along a path, whilst paying close attention to any possible incidences of congestion.

The slow start algorithm works hand-in-hand with the sender's *cwnd* variable introduced earlier, whose size is typically maintained in bytes. Upon a new TCP connection being initiated, the algorithm sets the value of *cwnd* to one segment size, allowing the sender to send a maximum of one data segment into the network. When the sender receives an ACK signalling a positive transmission, it increments the size of its *cwnd* by one segment size, i.e. the *cwnd* size is now equal to two. The sender can now send out a maximum of two data segments in succession into the network, for which it will receive two further concurrent ACKs. Again, for each ACK the sender receives it increments the size of the *cwnd* by one segment size, i.e. the *cwnd* size is now equal to four. This process continues, resulting in an exponential growth of the size of the *cwnd*.

It should be noted that in each instance during the connection, a sender is only permitted to send a number of data segments up to the minimum of its *cwnd* and the receiving host's advertised receive window, *rwnd*, as dictated by Eq. 1.1.

The slow start algorithm can still be quite aggressive if allowed to continue without limits. Essentially it is only used to get the *cwnd* size up to a point where network tolerance has been reached, a point where any further aggressive injections of data could cause congestion in the network path. Hence, the *ssthresh* threshold variable is maintained by the TCP sender, whose value determines the point at which the slow start algorithm should terminate based on the size of the *cwnd*. When the size of the *cwnd* exceeds the value of *ssthresh* the sender then switches to the congestion avoidance algorithm in order to continue the sending of data, albeit in a less aggressive manner to maintain network stability.

The *ssthresh* is continuously and dynamically updated throughout the life of a TCP connection, as it is an important indicator of conditions in the network. Typically, its initial value is set to 65535 bytes. Details of how the value of *ssthresh* is varied will be discussed in the sections to follow.

2.2.2 Congestion Avoidance

Congestion avoidance is the algorithm used by TCP senders once network tolerance has been reached. Its duty is to maintain a steady state and stable share of the available resources [93]. In the stable state a TCP connection is in a state of equilibrium, where the sender is injecting new data segments into the network at the same rate at which it is receiving ACKs [49]. The algorithm operates on the premise that the network may allow a little more data to be injected into it; the objective being to utilise any additional bandwidth along the path, if available, whilst remaining fair to other TCP connections that may be sharing the same network path. Since it is operating at equilibrium, the sender should be able to dynamically adapt to sudden changes in the condition of the network path.

Whilst in the congestion avoidance phase a TCP sender makes use of free network resources by controlling the size of the same *cwnd* variable used in the slow start phase, whose value is now greater than *ssthresh*. The sender now increases the size of the *cwnd* by $1/cwnd$ on receiving every new ACK for sent data. This technique ensures a slow linear growth of the *cwnd*, allowing the sender to gently probe for any

additional bandwidth that can be utilised. Again, the sender is only permitted to send a number of data segments as dictated by Eq. 1.1.

The sender remains in the congestion avoidance phase until it detects the loss of a segment between the two communicating end-hosts, signalling to TCP that congestion has occurred somewhere in the network path. TCP should now react accordingly to alleviate the imminence of congestion, preventing further segment losses.

2.2.3 Loss Detection in TCP

As documented in [99], a TCP sender is able to detect the loss of a segment in one of two ways; i) through the reception of three duplicate ACKs, or ii) the expiration of a retransmission timer.

2.2.3.1 Duplicate Acknowledgements (DUPACKs)

A duplicate acknowledgement (DUPACK) is the name given to ACKs that have already arrived at the sender, thereby providing no new insights. A DUPACK is identical to a regular ACK in the sense that they both acknowledge the same sequence of data bytes received by indicating the same next expected sequence number.

DUPACKs are generated in response to new data segments arriving out-of-order at the receiver, because TCP aims to guarantee an in-order delivery of data to receiving applications. Therefore it also expects to receive data segments in-order from a sender.

If then a data segment is dropped (or lost) somewhere in the network path on its way to receiver, then the next data segment in sequence will arrive at the receiver. However, because the receiver was not expecting this particular segment, it generates another ACK for the expected segment. This is called a DUPACK. The receiver will continue to generate DUPACKs for each unexpected data segment it receives, until eventually the expected in-sequence segment arrives, which will generate a new ACK. Note that DUPACKs are also generated when data segments arrive out-of-order at the receiver due to taking different paths through the network, which is a different situation to a completely lost segment. The sender should therefore be able to detect the loss of a segment from the DUPACKs it receives.

Using the arrival of DUPACKs, a TCP sender assumes that a sent data segment has been lost when three DUPACKs arrive in succession. Upon receiving three DUPACKs the sender no longer assumes that the expected segment may have been delayed in the network due to re-routing. It therefore takes immediate action by invoking the fast retransmit algorithm to handle the retransmission of data to resolve the missing segment (or sequence of bytes). Figure 2.2 illustrates the exchange of data segments and ACKs between a sender and a receiver pair, illustrating the growth of the *cwnd* size as well as the idea of segment retransmissions due to triple DUPACKs arriving at the sender.

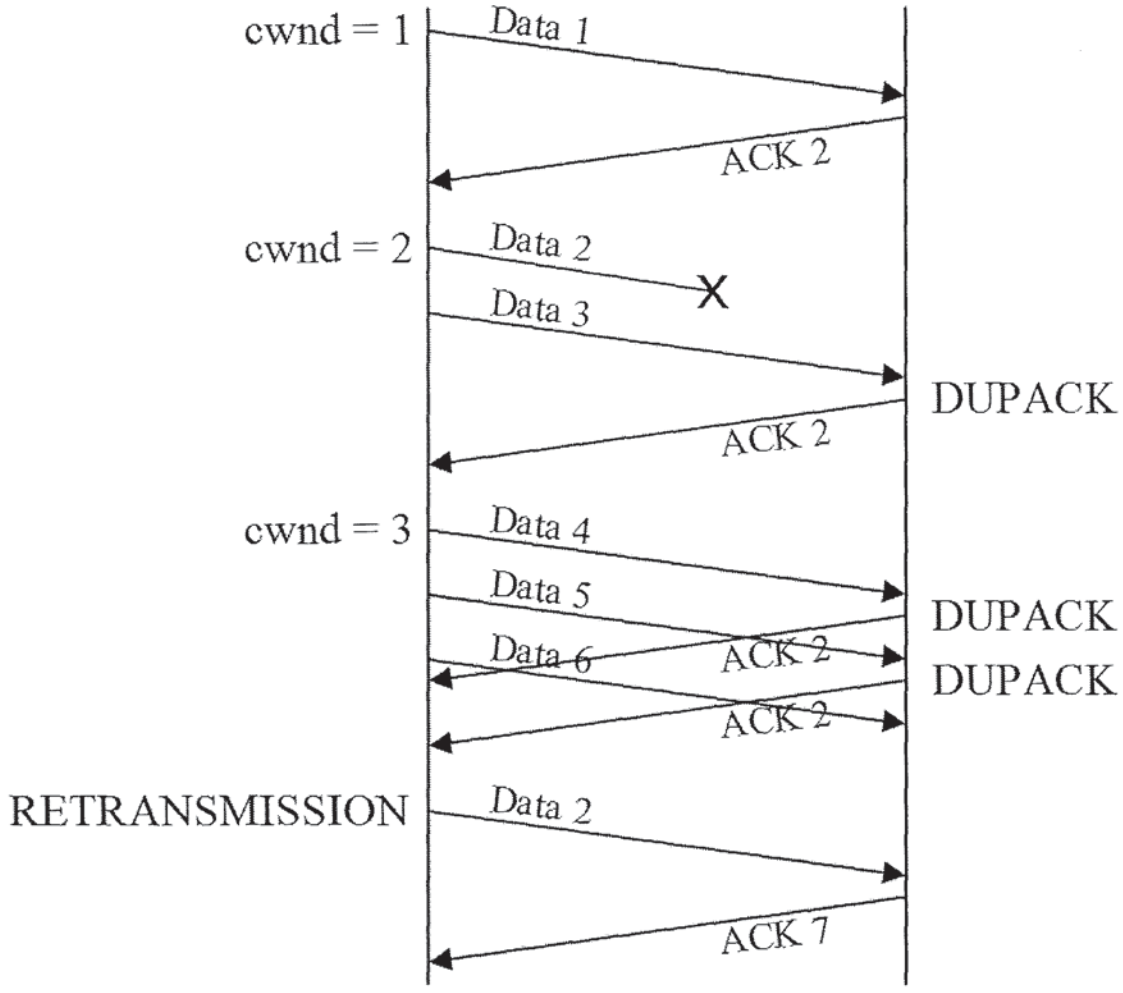


Figure 2.2: The concept of ACKs, DUPACKs, *cwnd* size evolution, and retransmissions

2.2.3.2 Retransmission Timeout (RTO)

A TCP sender initiates and maintains a retransmission timer linked with the transmission of every data segment. The timer is reset when all outstanding sent data

segments have been successfully acknowledged. However, if the ACKs do not arrive within a certain time frame then the timer will expire. This event is known as a *retransmission timeout* (RTO) at the sender [49]. A RTO event leads TCP to infer that the segment in concern has been completely lost somewhere in the network path because the receiver did not generate a single ACK for it within an appropriate time-frame. The TCP sender reacts by immediately retransmitting the earliest segment that has not been acknowledged. At the moment prior to the retransmission, the sender sets the value of the *ssthresh* variable to *cwnd*/2, and then sets the size of the *cwnd* to one segment size. It then reverts to the slow start algorithm with the *cwnd* size equal to one, because a RTO event is regarded as a case of serious network congestion by TCP [49]. Eventually, when the *cwnd* size reaches the value of *ssthresh*, the congestion avoidance algorithm takes over. Note here that the *ssthresh* value acts much like a pointer, remembering the last *cwnd* size which caused the congestion, thus preventing it from occurring again.

There have been many studies relating to the appropriate computation of the RTO timer [104] [105] [106]. The idea is to manage the retransmission timer in such a way that a data segment is never retransmitted too early. However, the suggested technique that all TCP implementations must adhere to has been well documented in [107], which still stands (or holds) today. In a nutshell, the timer's value is a derivation of the currently measured and smoothed estimate of the RTT and its mean deviance, as originally proposed by Jacobson [91].

In summary then, when the RTO timer expires and after the earliest unacknowledged segment has been retransmitted, TCP senders must immediately double the value of the current RTO timer, and restart it with the new value for any further segment transmissions.

2.2.4 Fast Retransmit

As specified in [99], a TCP sender should retransmit data segments upon the detection of a loss. The fast retransmit algorithm was developed to provide a speedy mechanism for the retransmission of segments.

A sender invokes the fast retransmit algorithm upon the successive arrival of three DUPACKs, as discussed previously, leading to the immediate retransmission of what appears to be the missing segment without any further delays. The objective is to speed up the transfer without waiting for the RTO timer to expire. Each invocation of the fast retransmit algorithm leads to the retransmission of only a single data segment at a time. Immediately after the algorithm has performed the retransmission due to the reception of three DUPACKs it hands over responsibility of the *cwnd* and further transmissions to the *fast recovery* algorithm.

2.2.5 Fast Recovery

Fast recovery is an important algorithm that allows a TCP sender to maintain high sending rates in the presence of retransmissions due to segment losses, and where the network path may be only moderately congested. Because of the arrival of three DUPACKs that lead to the retransmission, there is some evidence that data segments must still be arriving at the receiver, therefore generating the DUPACKs. From this information the TCP sender can infer that data is still flowing along the network path, and hence the levels of congestion are not extreme [49].

At this point the sender reverts to the congestion avoidance algorithm, as opposed to slow start which would drastically reduce the flow of data and cause a significant drop in the sending rate. Also, because the *cwnd* may have grown to a large size, and if segment losses are an infrequent event, it would be wiser to resume the TCP connection from the congestion avoidance phase. The sender therefore sets the value of the *ssthresh* variable to *cwnd*/2, and then sets the size of the *cwnd* to *ssthresh* + *3 segment sizes*, ensuring the connection resumes in the congestion avoidance phase [99]. The addition of three segment sizes to the *cwnd* is to account for the three DUPACKs that have just left the network, which tells the sender that there is immediate capacity in the network for three segments [49].

Then each time another DUPACK arrives at the sender, it accounts for its departure from the network by incrementing the *cwnd* by one segment size. If it is allowed by the new size of the *cwnd*, a data segment is sent into the network by the sender. The idea is to keep data flowing between the two end-points, whilst waiting for lost segments to

be recovered. Finally, upon the arrival at the sender of a new ACK acknowledging new data, the sender deflates the size of the *cwnd* to the current *ssthresh* value, which reinitiates the congestion avoidance algorithm.

Figures 2.3 to 2.5 provide theoretical illustrations of the behaviour of the sender's *cwnd* in the event of no segment losses (Figure 2.3), upon the arrival of triple DUPACKs (Figure 2.4), and upon expiry of the RTO timer (Figure 2.5). All illustrations are based on TCP Reno, as documented in [99].

Figure 2.3 shows a perfect display of the slow start phase, followed immediately by the congestion avoidance phase when the *cwnd* is equal to 65 Kbytes. Figure 2.4 shows how the *cwnd* is affected when three DUPACKs arrive back-to-back between the 12th and 14th RTT of the plot. At this point the *cwnd* size is halved as per the fast retransmit algorithm, and then the fast recovery algorithm takes over at the 15th RTT. Eventually a new ACK arrives in the 18th RTT, which hands over control back to the congestion avoidance algorithm.

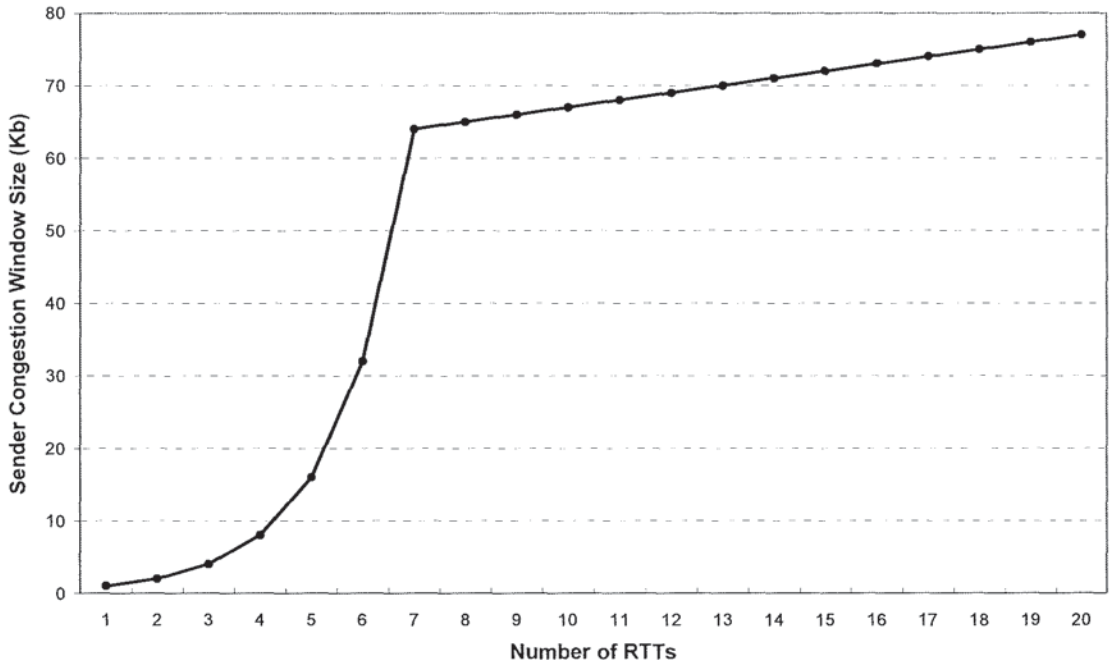


Figure 2.3: Evolution of TCP Reno sender's *cwnd* size under no losses (*ssthresh* = 65 Kb)

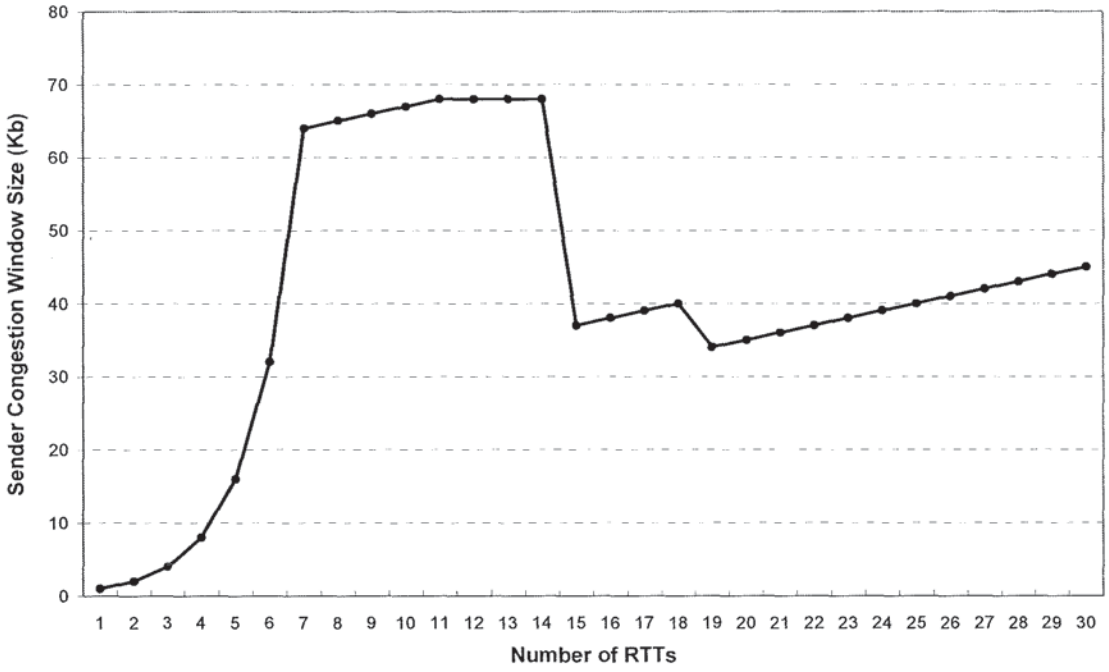


Figure 2.4: Evolution of TCP Reno sender's *cwnd* size under the reception of three DUPACKs

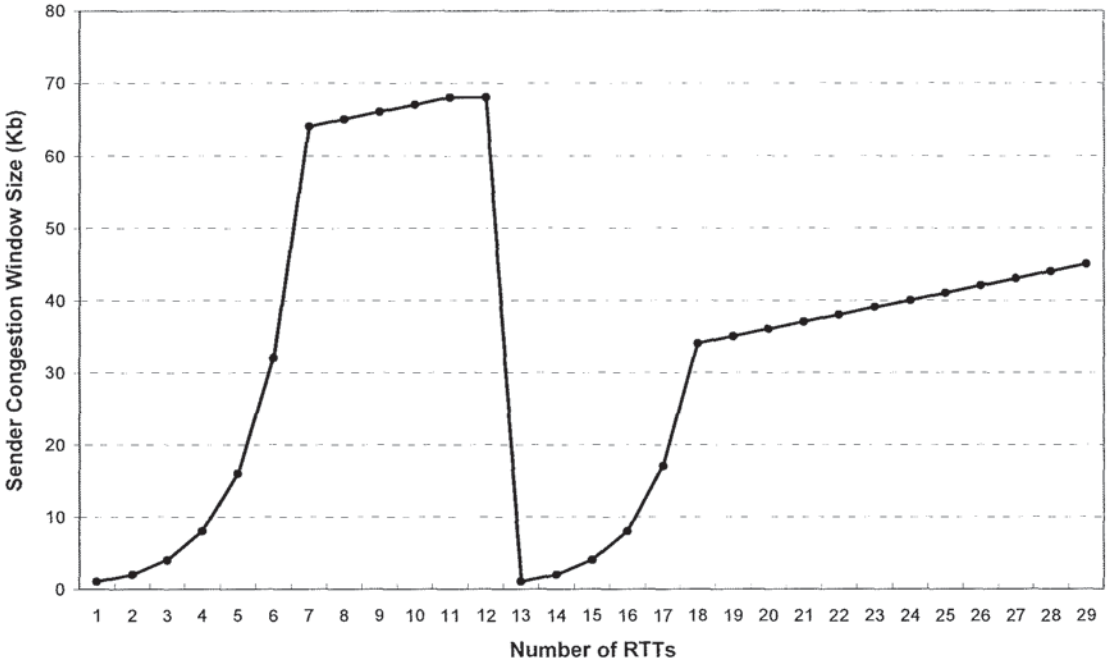


Figure 2.5: Evolution of TCP Reno sender's *cwnd* size under a retransmission timeout event

Figure 2.5 shows how the sender's *cwnd* is affected when there is an RTO event, which starts at the 11th RTT in the plot, whilst in the congestion avoidance phase. Shortly after the 12th RTT the sender's RTO timer expires due to the non-arrival of an ACK. The sender therefore retransmits the presumed lost data segment and reduces the

cwnd size to one segment before reinitiating the slow start algorithm, albeit with a lower *ssthresh* value.

2.2.6 Common TCP Features in the Modern-Day Internet

The first twenty-five years of the Internet consisted of TCP running over slower and smaller networks, supporting simple applications. In contrast, the last decade of the Internet has seen interconnected networks become faster and more heterogeneous, with applications becoming diverse and more demanding. Thus, as the Internet has evolved into a complex cloud, so has TCP [8]. Today, TCP supports a host of options and features (supplementary to its congestion control algorithms) that have been bolted-on by researchers over the years. These features should be used by TCP implementations in the Internet today [108].

This section therefore provides the reader with a brief overview of the most common TCP options and extensions that should be used by implementations today, many of which are documented in [109].

2.2.6.1 Selective Acknowledgements (SACK)

The first official enhancements to the original TCP [1] led to the release of TCP Tahoe, proposed in 1988, which incorporated the slow start, congestion avoidance, and fast retransmit algorithms [91].

Further modifications were made in 1990 to the congestion avoidance algorithm, as proposed in [93]. An additional fast recovery [99] algorithm was also introduced, which led to the release of another enhanced version of TCP, TCP Reno.

It was in 1995 when user-communities realised that further enhancements were needed to TCP Reno in order to cater for the fact that multiple segment losses from the same *cwnd* of transmissions were causing poor TCP throughputs. It was discovered that a TCP sender often had to wait for the expiration of the RTO timer in order to recover from multiple segment losses per *cwnd* of segments in flight [110]. This was due to the fact that TCP uses a system of cumulative acknowledgements, which forces the sender

to either wait for an entire RTT to find out about each lost packet, or to unnecessarily retransmit segments which have been correctly received [111].

Hence, the *selective acknowledgement* (SACK) option was proposed as a technique to correct this behaviour in the face of multiple segment losses per *cwnd* [112]. With the SACK option, the TCP receiver is able to 'selectively' inform the sender about all segments that have arrived successfully, so the sender only needs to retransmit those segments that have actually been lost.

The use of the SACK option for TCP connections requires support and participation from both senders and receivers. When used, the receiver utilises space in the header of ACK segments to convey information back to the sender about each contiguous block of data that has arrived. From this information, the sender is able to infer all missing data segments in a single RTT, without having to wait for a RTO.

2.2.6.2 Reno vs. NewReno

To overcome the fact that many TCP hosts at the time in the Internet did not support the use of the SACK option, or that certain hosts did not have a willingness to support such a feature, the problems caused by multiple segment losses from a single *cwnd* were still present. It was assumed that TCP Reno was the most widely used variant at the time, and hence any further modifications should be based on this. In 2004, slight, yet significant modifications were made to the fast recovery algorithm of TCP Reno to handle multiple losses per *cwnd*, which led to the release of *TCP NewReno* [113]. Its proposal was initially aimed at those TCP implementations that were unable to use the TCP SACK option at the time. Also, it is shown in [113] that TCP NewReno can improve the performance of the TCP Reno's fast recovery algorithm in a wide variety of scenarios. As a final note, according to [100], in 2003 some of the most widely deployed variants of TCP in the Internet were Reno, NewReno, and where feasible, Reno with the SACK option.

In summary, both TCP NewReno and TCP Reno with SACK are solutions for tackling the same problem.

2.2.6.3 Maximum Segment Size

The TCP maximum segment size (MSS) is an important consideration in connections that traverse the Internet. It is the largest amount of data, specified in bytes, which a host machine or communications device can process as a single unfragmented unit [114].

Generally, the number of bytes of data in a TCP segment plus the number of bytes making up the segment's header fields should not be greater than the *maximum transmission unit* (MTU) size of the lower layers in the TCP/IP network protocol stack. This is to prevent fragmentation of TCP segments by intermediate protocols and devices whilst in transit through the network path. Fragmentation can drastically slow down the speed of TCP connections due to increased overheads.

Typically, the MTU size in the Internet is 1500 bytes [49]. The TCP header size per segment requires 20 bytes, and the attached IP header per segment requires an additional 20 bytes. Today, this leaves TCP with a MSS of 1460 bytes per segment for actual data.

2.2.6.4 Window Scaling

The *window scale* extension expands the definition of the TCP window field from the legacy 16 bits to 32 bits using a scale factor [109]. It is applied to the windows of both the sender (*cwnd*) and the receiver (*rwnd*), as they both limit the throughput of a TCP connection (Eq. 1.1). A 16 bit window field allows a maximum window size of just 65,536 bytes, which is rather small in today's high-bandwidth, high-delay paths across the Internet [115]. A 32 bit window field increases the maximum possible window size significantly, for both the sender and the receiver.

The crucial scale factor is communicated by a sender in a new 3 byte TCP window scale option in the header of the first segment exchanged between a sender and receiver. This option is an offer, but not a guarantee. The receiver must also send back a window scale option in its first segment to the sender to enable window scaling up to 32 bits in both directions.

2.2.6.5 Timestamps

The *timestamps* option was suggested for providing greater precision in TCP's *round trip time measurement* (RTTM) mechanism, which provides more accurate estimates of the RTT experienced by data segments [109].

As stated in [90], accurate and current estimates of the RTT are necessary for TCP to be able to dynamically adapt to changing traffic conditions in network paths, a common characteristic of today's heterogeneous Internet [8]. This includes being able to maintain a conservative RTO timer value throughout the lifetime of TCP connections.

Legacy TCP senders based their RTT measurements upon a sample of only one segment per *cwnd*, resulting in poor RTT estimates when the *cwnd* size became larger with many segments being sent in succession, as is commonly the case in today's high-bandwidth, high-delay paths across the Internet [115].

Hence a solution to the problem is to use the TCP timestamps option field in segment headers. In a nutshell, a sender places a timestamp in each data segment, and the receiver reflects these timestamps back in its ACKs. Upon receiving the ACK, the sender performs a simple subtract operation giving the sender an accurate RTT measurement for every ACK received. This is the RTTM mechanism [109].

2.2.6.6 Delayed ACKs

The *delayed ACKs* option provides a TCP receiver with the option of increasing network efficiency by acknowledging every arriving data segment within a short time interval [88]. This leads to fewer ACKs being generated in the reverse path in relation to the number of data segments travelling in the forward path [116].

In accordance with the usage of delayed ACKs, a TCP sender must measure the effective RTT including the additional time due to ACKs being delayed, thereby preventing unnecessary retransmissions due to RTO events.

When using the delayed ACK option, a receiver cannot delay the generation of an ACK for more than 500 milliseconds, and when receiving full-sized TCP data segments it should generate an ACK for at least every second incoming segment [91].

2.3 IEEE 802.11 WLANs – An Overview

Wireless technologies are infringing on the traditional territory of 'fixed' or 'wired' networks. The world is becoming more mobile, and as a result, traditional methods of networking are proving inadequate to meet the challenges imposed by the networking world's collective lifestyle [23].

Any new wireless technologies targeted at computer networking must promise to do what traditional methods have always been capable of doing, and one such technology that promises this is the *802.11 Wireless LAN* (WLAN) standard from IEEE Working Group 11 [20]. This is the protocol that has been designed to provide an experience as much like that of wired Ethernet as possible.

The IEEE 802.11 WLAN standard offers one key advantage to its users, *mobility*. Wireless end-users can connect to existing backbone networks, such as the Internet, and can roam around freely away from the tethers of an Ethernet cable. 802.11 WLANs also possess the feature of being extremely *flexible*, which explains their rapid deployment capabilities and huge take-up trends in recent times [117]. Coupled with the unlicensed nature of the 2.4GHz *industrial, scientific, and medical* (ISM) band in which they operate [23], 802.11 WLANs have scaled very well, especially at the edges of the Internet. In fact, one of the biggest growth areas for the take-up of 802.11 WLANs has been the home/residential market by wireless broadband users [47].

In short, the 802.11 WLAN specifies an enhanced *medium access control* (MAC) and a new *physical layer* (PHY) specification for all devices wanting to adopt the technology [20]. The 802.11 MAC and PHY layers have been designed to operate in unison to provide seamless and wireless connectivity to higher-layer applications and protocols [23].

2.3.1 Typical WLAN Topology

IEEE 802.11 WLANs use the idea of a centralised *access point* (AP) to connect a group of wireless users to an existing wired backbone infrastructure, as illustrated in Figure 2.1. Wireless end-users connect and communicate with at least one central AP; one wired Internet connection to an AP can therefore accommodate many wireless hosts [23]. This type of scenario is typically described as an *infrastructure* mode WLAN, and is typical of most 802.11 WLANs deployments today within home and office environments [47].

In more detail, the *basic service set* (BSS) is the fundamental building block of the IEEE 802.11 architecture. Hence an infrastructure BSS is defined as a group of wireless devices that are under the direct control of a single coordination function, which is defined in Section 2.3.2.1. However, while all users in an infrastructure BSS can communicate directly with each other via the AP in theory, degradations of the transmission medium due to multipath fading effects, or interferences from nearby BSSs using the same radio medium, can cause some wireless users to appear 'hidden' from others [22], which will be explained in Section 3.3.3.

2.3.2 802.11 MAC

The primary function of the 802.11 MAC is to provide the core access control functions, supporting the usage of shared-medium PHYs for wireless connectivity. The 802.11 MAC layer is actually a sub-layer within the *data link* layer of the OSI reference model, and is designed to support the original 802.2 *logical link control* (LLC) sub-layer. This layout enables traditional 802.3 wired Ethernet networks to communicate with 802.11 wireless networks using a common language of LLC encapsulation [23]. In general terms, the 802.11 MAC sub-layer defines a set of rules to determine how the radio medium is accessed in order to send data, whilst leaving the actual details of transmission and reception to the lower PHY, i.e. it characterises access to the radio medium in a way common to the various wired 802.x standards. Figure 2.6 illustrates where the 802.11 MAC sub-layer sits within the data link layer of the five-layer TCP/IP protocol stack. It also shows where the 802.11 PHY falls.

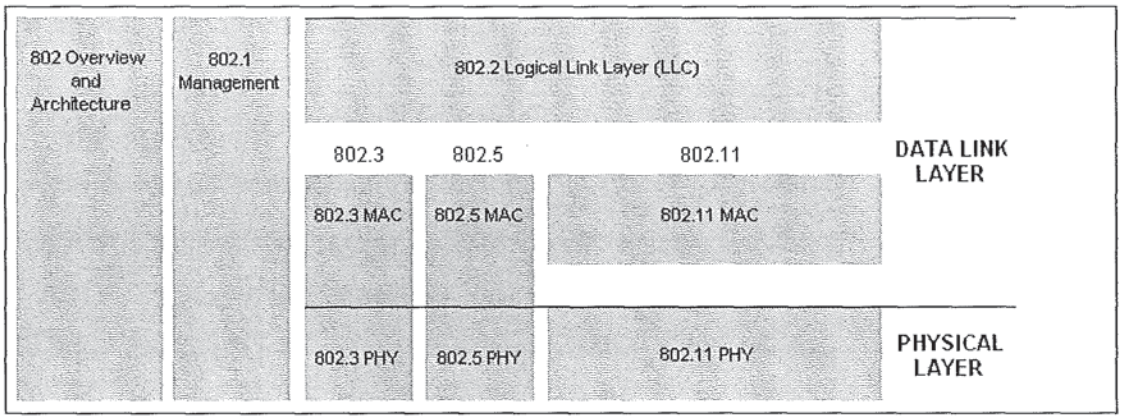


Figure 2.6: The IEEE 802 family

Of particular interest in the IEEE 802.11 specification is the support for two fundamentally different MAC schemes to transport asynchronous and time bounded services [22]. The first scheme, *distributed coordination function* (DCF), is similar to traditional legacy packet networks supporting best-effort delivery of data. The DCF is designed for asynchronous data transport, where all devices with data to transmit have an equally fair chance of accessing the transmission medium. The *point coordination function* (PCF) is the second MAC scheme, whose penetration did not really take-off in the marketplace [23]. The PCF is based on polling that is controlled by an access point (AP), and is primarily designed for the transmission of delay-sensitive traffic.

2.3.2.1 DCF Operating Mode (CSMA/CA)

In today's 802.11 WLAN deployments, the DCF operating mode remains as the most popular and successful MAC coordination function [118]. DCF is based on the *carrier sense multiple access with collision avoidance* (CSMA/CA) protocol, as opposed to the *carrier sense multiple access with collision detection* (CSMA/CD) protocol used in standard Ethernet systems [23]. Carrier sensing is performed by 802.11 devices at the air interface, referred to as *physical carrier sensing*, and at the MAC sub-layer, referred to as *virtual carrier sensing* [47]. The main difference between CSMA/CA and CSMA/CD is that with the former i.e. over wire media, transceivers have the ability to listen while transmitting and so can detect collisions. But, even if a radio transceiver could listen on the channel while transmitting, the strength of its own transmissions would mask all other signals on the air. So, the 802.11 DCF cannot

directly detect collisions as with wired media, and therefore takes great measures to avoid collisions.

Focussing on the CSMA/CA protocol, DCF uses a *random back-off timer* mechanism for preventing access to the radio medium by multiple wireless devices when the medium is in use by one of the devices [20]. Collisions are most likely to occur immediately after the medium becomes available for transmissions because multiple devices would have been waiting for just a single medium to become free. To prevent this all 802.11 devices randomly compute and initiate a back-off timer in the range 0 to 7 when the medium is sensed as busy. Upon sensing the medium as being idle, all devices begin to decrement their back-off timer. If the medium becomes busy again, the devices freeze their timer. Only when the time reaches zero a device gains exclusive access to the radio medium for transmissions. The random back-off arrangement is therefore used to resolve medium contention conflicts by promoting fairness amongst devices [47]. Fairness is maintained in the WLAN because all wireless users have an equal probability of gaining access to the transmission medium [53].

2.3.2.2 Transmission of Frames

There are three different types of frames supported by 802.11: *management*, *control*, and *data* frames. Management frames are used for functions such as managing associations and disassociations with the AP, as well as timing and synchronisations. Control frames are used for acknowledging successful transmissions. Data frames are used for the transmission of actual data [23].

All 802.11 frames consist of a *control field* that states the 802.11 protocol version, the frame type, and various other indicators, such as whether or not power management is being used, and so forth. In addition, all frames contain MAC addresses of the source and destination devices (including the AP), as well as an identifying *frame sequence number* (in the range 0 to 4095), a *frame check sequence* (FCS) (for error detection), and finally the *frame body* itself. In data frames the frame body encapsulates data from higher-layer protocols such as TCP [23]. Figure 2.7 illustrates the basic structure of an 802.11 frame.

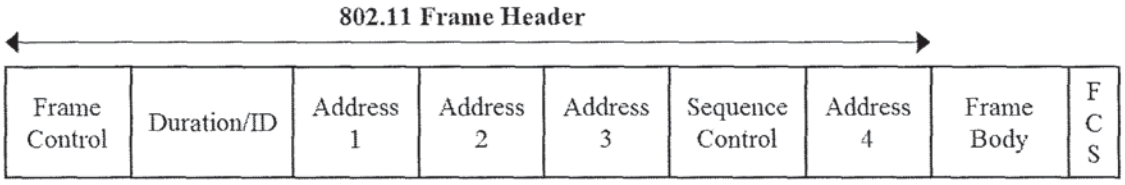


Figure 2.7: The structure of 802.11 data frames

The transmission of a data frame by the 802.11 MAC is initiated by the sensing of the transmission medium as idle. Upon successfully acquiring a transmission slot, it transmits the data frame. Upon successful arrival of the data frame, the receiving device transmits a DCF *positive acknowledgement* (802.11 ACK) frame back to the sender to indicate a successful transmission. The sender then progresses to the next data frame to be transmitted in sequence, and the process continues [47]. Frame sequence numbers are incremented by one for each new whole data frame transmission. Upon reaching 4095, they wrap around to zero and continue.

2.3.2.3 Error Detection and Recovery with DCF

Error detection and recovery of data frames is the responsibility of sending devices in 802.11 DCF WLANs [23]. Upon the transmission of a data frame, a sending MAC uses a 32 bit *cyclic redundancy check* (CRC) over the entire contents of each frame, generating a FCS value unique to that frame. The value is then placed in the FCS field of the frame prior to its transmission. The receiving device then performs an identical CRC calculation over the entire contents of the received frame (excluding the FCS field). It then compares its own FCS value with the received FCS to verify whether or not any errors occurred in the frame during its transmission. A successful FCS check by the MAC implies the received frame is intact and is therefore accepted and passed up to the higher-layers. An 802.11 ACK is then transmitted back to the sending device. Note that 802.11 ACKs do not contain any information regarding sequence numbers of data frames being positively acknowledged. It is the responsibility of a sending device to infer whether or not the transmission was a success [20].

With regards to error detection in DCF, a sending MAC relies on the arrival of 802.11 ACKs to confirm successful deliveries of data frames. It expects an 802.11 ACK for each and every data frame transmitted [23]. A sending MAC does not progress to the

next in-sequence data frame transmission until the current frame is positively acknowledged by the receiving MAC. The lack of reception of an expected ACK indicates to the sender that a transmission error has occurred; either the data frame did not arrive at the receiver, or it failed its FCS check at the receiver [20]. Note, however, that the receiving MAC may have received the data frame successfully, and that the error may have occurred in the transmission of the ACK on its way to the sender. Either way, to the sending MAC this situation is indistinguishable from an error occurring in the initial data frame transmission.

With regards to error recovery in DCF, a sending MAC utilises a *stop-and-wait automatic repeat request* (ARQ) mechanism to recover from transmission errors of data frames, i.e. it performs MAC level frame retransmissions locally over the WLAN [20].

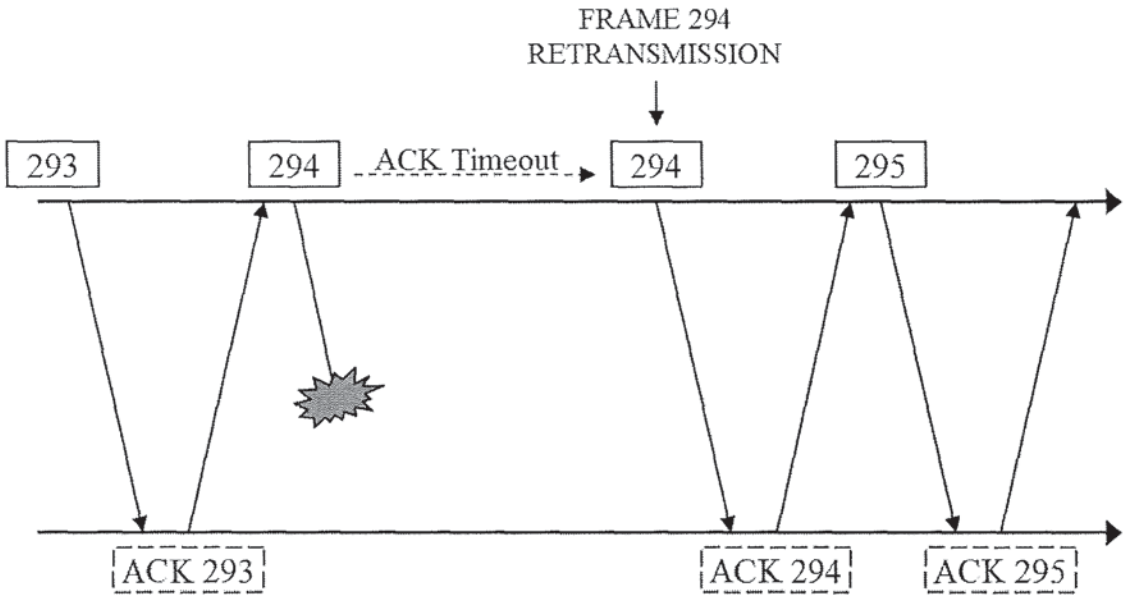


Figure 2.8: 802.11 frame exchange sequence diagram under a data frame loss

Upon detection of transmission errors, a sending MAC must retransmit data frames using the same frame sequence number, until positively ACKed. Note that each data frame has associated with it a *retry bit* in its header control field, which is set to 1 by the MAC when the frame is being retransmitted. A retransmission timer is used by the MAC to set a maximum amount of time to wait for the arrival of an 802.11 ACK [20].

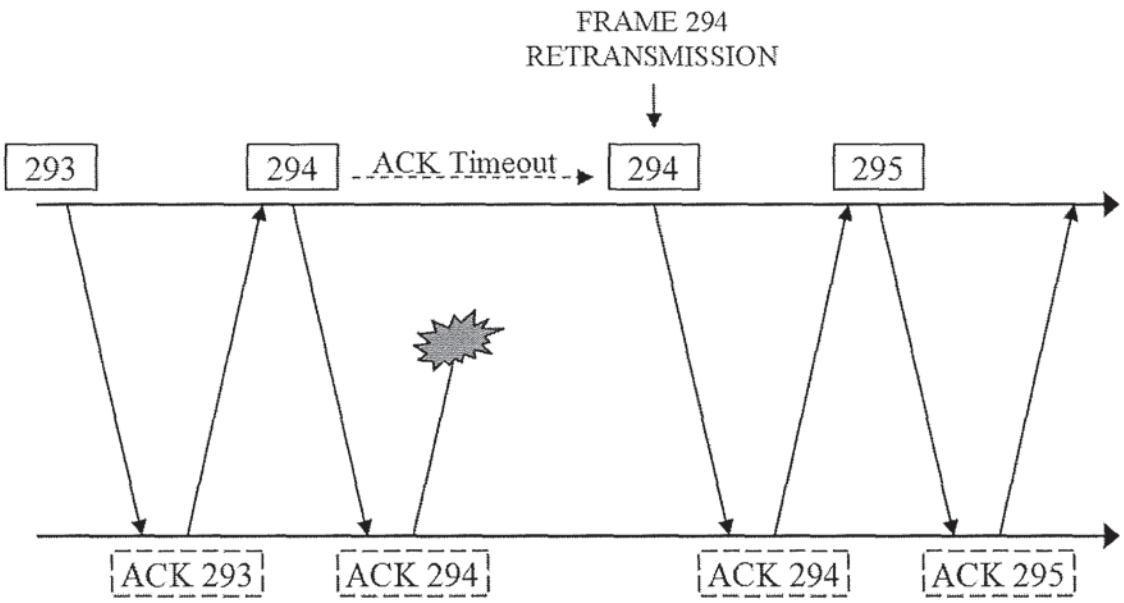


Figure 2.9: 802.11 frame exchange sequence diagram under an 802.11 ACK loss

Associated with data frame retransmissions is the *retry limit* that is maintained by the sending MAC for each individual incident. The retry limit of the MAC puts a limit on the maximum number of retransmission attempts it can make to recover a data frame. A counter beginning at zero is incremented for each retransmission attempt, and when the counter value reaches the retry limit the MAC discards the current data frame and makes no further attempts at retransmitting. It immediately moves onto the next in-sequence data frame from its send queue, resetting the counter's value to zero [23]. Figure 2.8 illustrates the sequence of exchanges between a sending MAC and receiving MAC under the loss of a data frame. As can be seen, a lost data frame is detected via an ACK timeout at the sending side, which results in a retransmission of the same data frame (with sequence number 294). The sending MAC continues with data frame 295 only after data frame 294 has been positively ACKed. Figure 2.9 illustrates the exchange sequence when an 802.11 ACK is lost in transmission. Again, the sending MAC relies on an ACK timeout to detect any errors, regardless of whether the loss occurred in the forward channel or reverse channel.

2.3.3 802.11 PHY

Like all networks, the transmission of data is always over a certain physical medium. In the case of wireless networks the medium is a form of electromagnetic radiation [25]. To be more precise, the medium used as the physical transmission layer (PHY) in

IEEE 802.11 WLANs are radiowaves allocated in the 2.4GHz ISM band [32]. IEEE 802.11 wireless devices can operate in this unlicensed band providing they adhere to certain requirements such as using a transmission power that is below a certain level, which varies from continent to continent [25].

The IEEE 802.11 standard specifies four different PHY channel modulation implementations for usage in the 2.4GHz spectra: *direct sequence spread spectrum* (DSSS), *orthogonal frequency division multiplexing* (OFDM), *frequency hopping spread spectrum* (FHSS), and *infrared* (IR), each of which are well documented in [119].

2.4 Multiple Standards

There are now in fact multiple standards under the IEEE 802.11 heading, since its original ratification in 1997 [22]. Whilst the all use the same enhanced MAC for DCF functionality, the standards differ in their implementations of the PHY technology used.

The original 802.11 standard of 1997 provided wireless users with data rates of up to 2 Mbps over the WLAN using a basic DSSS modulation system over the PHY, while the current and more successful IEEE 802.11b [120] (ratified in 1999) and IEEE 802.11g [121] (ratified in 2003) standards can offer data rates of up to 11 Mbps and 54 Mbps respectively. The 802.11b and 802.11g standards have experienced the most widespread deployment in today's WLANs due to their promising data rates [47].

The 802.11b standard achieves up to 11 Mbps by using an extension of the legacy DSSS modulation system, by using *complementary code keying* (CCK) over the PHY. The 802.11g standard is a further amendment to the 802.11 PHY, using the OFDM modulation system to provide higher data rates up to 54 Mbps over the WLAN. In addition, the 802.11g also supports the DSSS-CCK modulation system, allowing it to revert to the lower data rates of 802.11b, thus making 802.11g devices backward compatible with 802.11b devices [20]. Figure 2.10 illustrates the multiple PHY variants that make up the multiple IEEE 802.11 standards using a common MAC protocol.

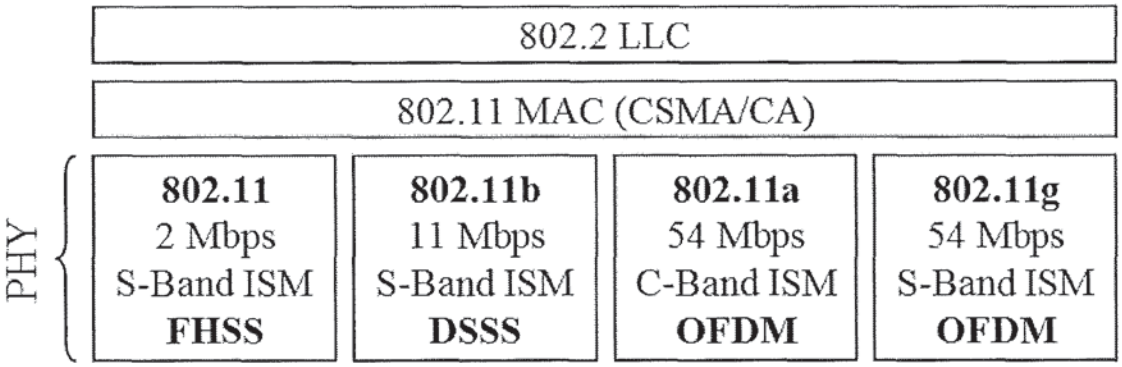


Figure 2.10: The 802.11 PHY family with a common MAC

2.4.1 IEEE 802.11b

The 802.11b standard supports the four different data rates over the WLAN using a combination of channel coding schemes derived from DSSS and CCK, as shown in Table 2.1.

Modulation Scheme	Maximum Supported Data Rate (Mbps)
Differential Binary Phase Shift Keying (DBPSK)	1
Differential Quadrature Phase Shift Keying (DQPSK)	2
Differential Quadrature Phase Shift Keying (DQPSK)	5.5
Differential Quadrature Phase Shift Keying (DQPSK)	11

Table 2.1: The supported data rates of the IEEE 802.11b WLAN standard

2.4.2 IEEE 802.11g

Due to the early maturity of the 802.11b standard, and the continued demand for higher data rates by end-users, the 802.11g standard supports a range of higher data rates using various OFDM modulations and coding schemes over the PHY, as shown in Table 2.2.

Modulation Scheme	Maximum Supported Data Rate (Mbps)
Binary Phase Shift Keying (BPSK)	6
Binary Phase Shift Keying (BPSK)	9
Quadrature Phase Shift Keying (QPSK)	12
Quadrature Phase Shift Keying (QPSK)	18
16 Quadrature Amplitude Modulation (16-QAM)	24
16 Quadrature Amplitude Modulation (16-QAM)	36
64 Quadrature Amplitude Modulation (64-QAM)	48
64 Quadrature Amplitude Modulation (64-QAM)	54

Table 2.2: The supported data rates of the IEEE 802.11g WLAN standard

2.5 Radio Channels in 802.11 WLANs

Unlike wired channels that are stationary and predictable, radiowave channels used by WLANs are extremely random and do not offer predictability. The mechanics behind radiowave propagation is diverse, but can be generally attributed to *reflection*, *diffraction*, *multipath fading*, and *path loss*, which all degrade the quality of wireless signals and transmissions [25]. Diffractions occur when there is no direct line-of-sight in the signal path between a transmitter and receiver. Reflections occur when there are multiple obstacles in the signal path between a transmitter and a receiver. Multipath fading occurs due to radiowaves taking different paths between a transmitter and receiver, causing signal degradations due to *destructive interference* at the receiver. Path loss occurs when the strength of the radiowaves decrease as the distance between a transmitter and a receiver increases, lowering the quality of a received signal [16].

With regards to radio channels for 802.11 WLANs, the 802.11 working group has documented the use of the 2.4 GHz band for WLAN devices. It specifies that the 2.4 GHz band should be divided into 14 separate radio channels, each of which can be used independently for transmissions. Unfortunately, because the 802.11 PHY requires a 25 MHz separation between channels, adjacent channels overlap and will interfere

with each other. For 802.11b and 802.11g devices, the UK allows unlicensed usage of the non-overlapping channels 1, 6, and 11 in the 2.4 GHz band [47].

2.5.1 Interference and Noise in Indoor Environments

There are various sources of *interference* and *noise* that are of concern to the quality of WLAN signals and performance. Noise is defined as an undesired disturbance within the frequency band of interest from third-party sources [119]. The disturbance can affect a radio signal by distorting the information being carried, and even block the signal completely. Interference is defined as the interaction between radiowaves from separate sources, leading to undesired effects to the original radiowave signal. Typically 802.11 WLANs are deployed inside buildings, and are therefore subjected to degradation from interference and noise due to the intrinsic characteristics of *indoor* radiowave propagation [47].

The basic sources include *adjacent channel* and *co-channel* interferences from nearby WLANs in densely populated deployments, and from other devices operating in the 2.4 GHz ISM band [47]. Bluetooth devices have also been shown to interfere when they coexist with 802.11 WLANs [122]. Microwave ovens operate in the ISM band also, and produce high levels of noise when in use. In certain scenarios, a microwave oven can completely block out the signal between a WLAN device and the AP. Overall, the popularity of the ISM band for unlicensed wireless communications has led to an overcrowding of consumer devices transmitting on nearby frequencies and radio channels, especially within home/office environments [47]. Hence, the effects of interference and noise in 802.11 WLANs are a common expectation amongst wireless end-users today.

Finally, radiowave propagation inside buildings (indoors) differ from traditional radio channels; a) the distances covered are much smaller, and b) the variability of the environment is much greater in relation to shorter transmitter-receiver separations. It has been observed that radiowave propagation within buildings is strongly influenced by the layout of the building, the construction materials used, and the type of building [25].

2.5.2 Measuring Signal Quality

The strength of the transmitted signals in radiowaves is often measured in the unit of power, *Watts* (W), or, in the case of 802.11 WLANs, *milliWatts* (mW). Because the power detected at the receiving end of transmissions can be several orders of magnitude smaller than the transmission power, a simple solution was to measure the received signal strength with the *decibel-mW* (dBm) unit. The dBm value can be easily derived from the mW measurement [119].

Researchers commonly use the term *signal-to-noise ratio* (SNR) to describe the quality of a received signal in WLANs. The SNR is simply the ratio between the measured signal power (in dBm) and the corrupting noise power (in dBm) [119]. It is usually quoted in *decibel* (dB) units. Today a receiver's SNR (dB) measurement of channel conditions in a WLAN has become the common standard for describing the quality of the radio channel path between itself and the AP.

Chapter 3

Related Work

Introduction

The optimisation of TCP for wired-to-wireless paths has been an area of extensive research in recent years, mainly due to the popularity of wireless communications today and the demand for becoming 'mobile' whilst remaining connected to TCP servers in the wired Internet. Secondly, TCP code has made its way into the networking stack of virtually every mobile and wireless device manufactured to date, reaffirming its dominance as the de-facto transport protocol in the wireless arena.

Unfortunately, it has all happened too quickly for TCP, because it was never designed to be used over networks that use radiowaves as a transmission medium. Today, TCP is subjected to a diverse spectrum of wired-to-wireless environments, ranging from satellite links, to cellular networks, and last-hop WLANs. Each type of scenario brings with it a set of challenges for TCP. In this chapter, the common set of challenges for TCP over wired-to-wireless paths are reviewed, based on findings from previous related studies.

The chapter then moves onto the problems faced by TCP when used in conjunction with IEEE 802.11 WLANs. Initially the key characteristics of 802.11 WLAN indoor radio channels are reviewed, leading to a discussion of the performance implications this has on a fixed TCP sender, which have been previously highlighted. Again, all this is based on a synthesis and findings from previous studies, which have been analysed and logically presented for the reader.

To conclude the chapter, the thesis presents an up-to-date review of some of the most prominent work and sender-side enhancement proposals in the arena of TCP optimisations for wired-to-wireless environments, broken down by the type of approach taken and then the solution.

3.1 End-to-End TCP in Wired-to-Wireless Environments

This section aims to summarise what are believed to be the fundamental and key issues for TCP when used in a traditional end-to-end manner over wired-to-wireless environments.

3.1.1 Wired-to-Wireless Paths for TCP

Using wireless links as a transmission path for TCP poses several fundamental challenges for it, since TCP was never designed for usage over unreliable links [13]. Radiowaves suffer from a number of propagation characteristics that make them unpredictable and unsuited for wired protocols like TCP, which rely on path stability, relatively lower error rates, and where losses occur mainly due to network congestion [16]. This is where TCP's congestion control algorithms come into the equation, designed entirely for dealing with such issues relating to network congestion [3].

Recall that TCP is a reliable end-to-end protocol, implying that it operates transparently over a communication path between a wired sender and a wireless receiver. This means that a wired sender in the Internet has no knowledge or awareness of conditions along the path, and is therefore solely dependent on the inter-arrival of cumulative ACKs and the sampling of RTTs to estimate path conditions. In traditional wired networks the end-to-end RTTs for TCP segments remained relatively stable, with low variability [15]. Secondly, segment losses were mainly due to buffer overflows at intermediate routers, which the TCP congestion control algorithms could quickly and efficiently recover from. Hence, TCP has been designed to be reactive in the presence of congestion related issues.

Today, there exists a wireless path alongside the traditional wired path for TCP senders to deal with. This means that segments are now subjected to the characteristics of radiowaves too, which include a higher rate of non-congestion related losses [11], higher variability in the propagation delay (again non-congestion related) [27], and greater variability in the available bandwidth over the wireless portion of the communication path.

3.1.2 Inappropriate Reductions of the Sender Congestion Window

It is well known that loss rates in wireless networks are generally higher than in wired networks [123] (refer to Table 3.1). This is because radio channels possess higher *bit error rates* (BERs), which corrupt the individual bits of packets travelling through the medium [124]. A single bit error in a wireless packet is enough to deem the entire packet as unacceptable upon arrival when *forward error correction* (FEC) is not used [124]. BERs also occur randomly and in bursts, potentially affecting an entire sequence of wireless packets [13] [70].

	Typical Bit Error Rate (BER)
WIRED NETWORKS	$< 10^{-12}$
WIRELESS CHANNELS	$\sim 10^{-6}$

Table 3.1: Comparison of BERs between wired networks and wireless channels [Source: [117]]

Unfortunately, TCP was not originally designed with such heterogeneity in mind, and instead has been tuned well to treat all segment losses as a sign of network congestion somewhere in the communication path. Because segment losses in traditional wired networks are quite infrequent [15], it was acceptable for legacy TCP senders to retransmit the segment and reduce their sending rate each time losses occurred. The idea is to alleviate congestion at intermediate routers by backing-off the injection rate of new data. TCP senders reduce their sending rate by drastically reducing the size of the *cwnd* upon the detection of each loss via the congestion control algorithms.

A TCP sender is therefore unable to discriminate between segment losses occurring in the wireless link and segment losses occurring in the wired link. In both cases, it reacts by invoking anti-congestion procedures, causing end-to-end throughput to drop. Although retransmitting the lost segment is the correct action to take by TCP, the issue lies with the fact that after each retransmission TCP also activates the congestion avoidance algorithm. Constant losses over the wireless link due to higher BERs lead to constant and inappropriate reductions of the sender's *cwnd*, ultimately leading to sub-optimal overall throughputs for wireless end-users of TCP applications [125]. Losses over the wireless link do not necessarily require a reduction in the sending rate of new

data by TCP senders, as wireless networks often continue transmitting data at the same rate in the presence of errors and losses, usually relying on local error recovery mechanisms to conceal the errors from higher-layer protocols such as TCP [15].

3.1.3 Non-Congestion Related Delays

In general, wireless links impose longer latency delays than purely wired links [126], which in turn increases the end-to-end RTT of TCP data segments in wired-to-wireless paths [27]. This affects a TCP sender's transmission rate because it relies on the measured RTT to advance the size of its *cwnd*. It has been shown that connections with longer RTTs have lower throughput levels because incoming ACKs take longer to arrive. Since the *cwnd* evolution is *ACK-clocked*, differences in the RTT can lead to different growth rates of the *cwnd* [127]. Figure 3.1 clearly illustrates the impacts on the growth rate of a sender's *cwnd* size when the end-to-end RTT is larger.

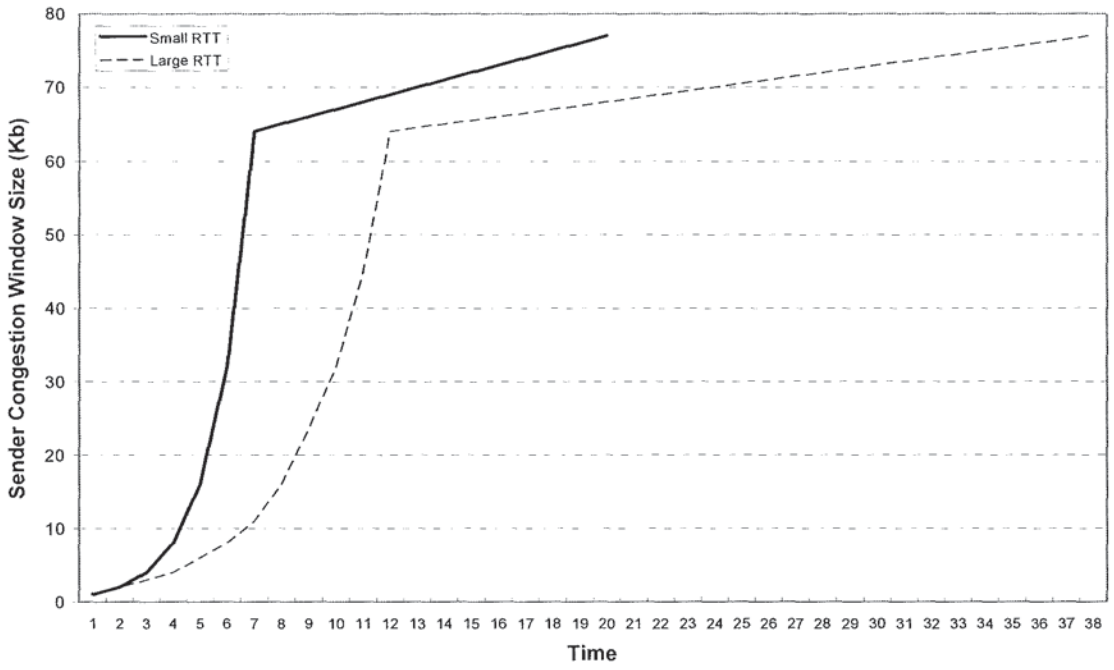


Figure 3.1: Differences in the evolution of a sender's *cwnd* size for different paths RTTs

Wireless networks also suffer from frequent temporal disconnections of mobile devices from the underlying network. Such disconnections are due to mobility of end-users within a wireless network, where a device is moved completely of the signal coverage range of the wireless gateway/controller, or due to 'black-outs' that often

occur in radio channels due to effects of shadowing from urban obstacles [11]. During a temporal disconnection, all wireless packets are dropped, affecting both TCP segments and ACKs. This can cause a TCP sender's RTO timer to expire, causing it to retransmit the dropped segments, and reduce its *cwnd* size to activate the slow start algorithm. If the connection is still in state of disconnection, retransmitted segments are also dropped, which leads to further RTO incidences and retransmissions at the sender. Such consecutive attempts at retransmissions due to timeouts will exponentially increase the RTO timer value according to Karn's back-off algorithm for each unsuccessful attempt [49]. This could potentially stall a TCP connection for many hundreds of milliseconds, even after a reconnection of the wireless device, as shown in Figure 3.2. Figure 3.2 illustrates the effects on a sender's *cwnd* growth when a TCP connection is stalled for several seconds (between time = 7 to time = 11) due to an inflated RTO timer,

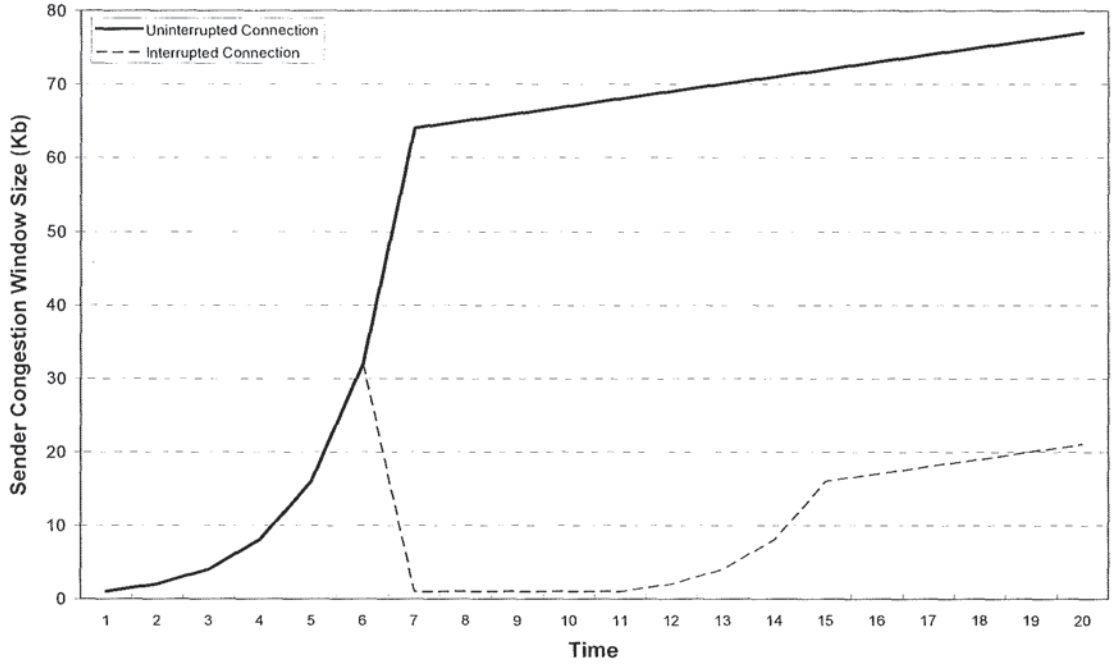


Figure 3.2: Highlighting the impacts of a large RTO timer on a sender's *cwnd* size evolution

Wireless networks often use local error recovery techniques such as ARQ and FEC to recover lost packets over the air, which often have high latencies associated with their usage. Whereas FEC introduces a relatively constant coding delay, ARQ techniques are more persistent with their retransmission attempts, which can translate into higher end-to-end RTTs for a TCP sender in the wired domain [15]. RTO events are a

common problem at TCP senders, brought on by sudden increases in the end-to-end delay when an ARQ mechanism is attempting to recover a lost packet locally. If an ARQ function is too persistent, a TCP sender will retransmit the unacknowledged segment after a timeout of the RTO timer. It is clear from this that whilst the retransmitted segment is on its way to the wireless domain, the wireless ARQ may have been successful with its recovery. Unfortunately TCP will have already unnecessarily reduced its *cwnd* size and sending rate due to a wasted retransmission effort, also wasting network resources and reducing throughputs for end-users in the wireless network.

3.1.4 Link Asymmetry Issues

A network communication path exhibits asymmetry with regards to TCP performance if the bandwidth in one direction is greater than the bandwidth in the reverse direction [128]. In wireless networks where a single gateway/controller typically services multiple wireless devices, the downlink (forward) channel and uplink (reverse) channel usually have differing capacities. Typically, the downlink radio channel capacity is configured to be greater than that of the uplink [11].

The key issue with such link asymmetries for TCP performance is that it results in a poor utilisation of downlink bandwidth because of the slower arrival of ACKs due to the limited uplink bandwidth [129]. This is because the TCP sender's *cwnd* is ACK-clocked, relying on the arrival of ACKs to advance the sending rate.

A further issue related to limited uplink bandwidths in wireless networks is where TCP ACKs are generated by end-devices at a rate faster than what can be actually sent back to the sender. This leads to the queuing of ACKs behind one another at a bottleneck, which can overflow. If ACKs get dropped then a TCP sender will timeout, thereby retransmitting the presumed lost segment and cutting back its sending rate. Hence a TCP sender can mistakenly interpret weak reverse channel conditions as simply congestion on the forward channel [11].

Another problem for TCP senders in the wired domain associated with wireless link asymmetry is caused by the *ACK-compression* effect [11]. Although the arrival of

ACKs at the sender will trigger an increase in the size of its *cwnd* and further transmission of new data segments, a sudden influx of successive ACKs from a reverse channel queue can break the sender's self-clocking behaviour. When this happens it can cause the sender to inflate its *cwnd* significantly, sending out a large burst of data segments into the network. This could burden the forward channel with an instant load, leading to congestion related issues.

3.2 Characteristics of Indoor 802.11 WLAN Channels

This section of the chapter reviews what is believed to be the typical characteristics of IEEE 802.11 WLANs when deployed indoors and in the infrastructure mode.

3.3.1 Frame Transmission Errors and Losses

Indoor 802.11 radio channels operating in the 2.4 GHz ISM band are prone to the effects of interference and noise from other in-building radio sources, as well as to the effects of path-loss and multipath fading due to mobility of end-user devices and from human-related obstacles [71]. This has a direct impact on the SNR measured by devices, which decreases when a receiving device is unable to distinguish the transmitted signal from the interference and noise levels [130] [131]. Consequently, the BER over the channel increases and can lead to random corruptions of 802.11 data frames and 802.11 ACKs, both of which translate into multiple retransmissions by the MAC, and even complete losses of data [76].

Hence an understanding of the behaviour of frame transmission errors in 802.11 WLANs has been an active area of great interest by researchers in recent years [70] [73] [74] [132] [133]. The aim of such studies is to be able to capture and model the transmission error probabilities of the wireless link, providing detailed insights into the workings of the ARQ mechanism in the DCF MAC. Although frame errors in 802.11 WLANs can be recovered locally by the transmitting MAC using ARQ retransmissions [50] [134], complete frame losses do still occur when the ARQ reaches its retry limit, as has been shown in [130] [135].

It has also been shown that frame errors in 802.11 WLANs can be caused by either random bit corruptions due to poor channel quality, or they can be caused due to colliding transmissions of frames, which are a normal operating feature of 802.11 WLANs according to [75]. Frame collisions occur only when more than one 802.11 device sends a data frame in the same medium access slot, as governed by the back-off timer of CSMA/CA protocol of the DCF at the MAC sub-layer [136].

3.3.2 Variable Frame Transmission Delays

The time taken for an 802.11 data frame to be transmitted through successful acquisition of the wireless medium and for it to be positively acknowledged by the receiving device is referred to as the end-to-end transmission delay of a data frame over the WLAN. The total delay of a data frame consists of the time spent in the transmission queue plus the time taken by the MAC to successfully transmit the frame, referred to as the MAC delay. The MAC delay consists of the channel access delay, which is the time spent by a device contending for access to the wireless channel, and the transmission delay, which is the time taken for a successful delivery including the acknowledgement [137].

The issues for higher-layer protocols that operate over 802.11 WLANs are related to the unpredictability of radio channel conditions, where frame errors and losses occur randomly, which therefore brings some form of uncertainty to the time taken to successfully deliver higher-layer data to an 802.11 end-user device. The uncertainty in end-to-end delays of data frame transmissions is because of the way in which the CSMA/CA mechanism is implemented in the MAC DCF, in conjunction with the ARQ mechanism. Initially, a device experiences a random back-off counter delay in the range 0 to CW before gaining access to the medium, where CW is the *contention window* size used by the MAC. Unfortunately, if a data frame is not delivered successfully (either due to a collision or poor channel conditions), then the sending MAC detects an error via an 802.11 *ACK-timeout*. Upon an ACK-timeout it reschedules a retransmission of the same data frame by contending for the medium once again using a new back-off counter interval, whose value is now chosen randomly from an increased range that is doubled, i.e. CW is doubled. After another unsuccessful transmission, the MAC reschedules another retransmission, again

doubling the range from which the back-off counter value is chosen at random. The sending device will keep retransmitting in this way until it reaches the retry limit of the MAC. To prevent excessively large contention delays per retransmission, the value of the back-off counter is limited by placing a cap on the maximum size of the CW, regardless of the number of attempts [138].

It is clear from the above procedures that the end-to-end transmission delay of any particular data frame over an 802.11 WLAN is potentially variable and unpredictable. It is a challenging task to accurately predict exactly how many retransmission attempts a data frame may experience, or the magnitude of the contention delays due to the randomness of the back-off timer. Fluctuating channel conditions can also influence the number of retransmissions that take place per data frame, which can considerably hike end-to-end transmission delays over the WLAN, and ultimately on global delays for higher-layer protocols. It has also been shown and proven that increasing the number of devices in an 802.11 WLAN using DCF CSMA/CA has a negative impact on the delay experience by any particular device contending for the wireless medium [53] [133] [139].

Many further studies have been undertaken to analyse and better understand the variability of frame transmission delays over 802.11 WLANs [52] [76] [134].

In [135] the authors highlight through experimentation and raise awareness of variable transmission delays associated with transmission queues building-up at the MAC sub-layer. In situations where higher-layer protocols have large streams of data to send over the WLAN, the 802.11 MAC will experience large queues of data frames. If the MAC is unable to service the queue quickly enough (due to continual frame errors occurring over the WLAN and retransmissions) then overall global delays for higher-layer protocols will increase due to the exacerbated effect of incoming data growing the queue size.

In [140] the authors also discovered via experimental work that there are additional transmission latencies associated with speed-switching of 802.11 hardware devices and software implementations, alongside those related to frame retransmissions.

3.3.3 Frame Collisions

An inherent problem in 802.11 WLANs is that of *collisions* amongst frames over-the-air, regardless of the CSMA/CA protocol used by the DCF. A collision immediately leads to the loss of all frames involved, and because there is no direct way to detect frame collisions, it can become a performance related issue [141]. In fact studies have shown that frame collisions in an infrastructure mode 802.11 WLAN can account for a greater percentage of all frame errors that occur [142].

The most commonly known cause of frame collisions is due to the *hidden node problem*. It occurs when a node (or 802.11 device) is visible by the AP, but not by the other nodes who are also in the range of the AP.

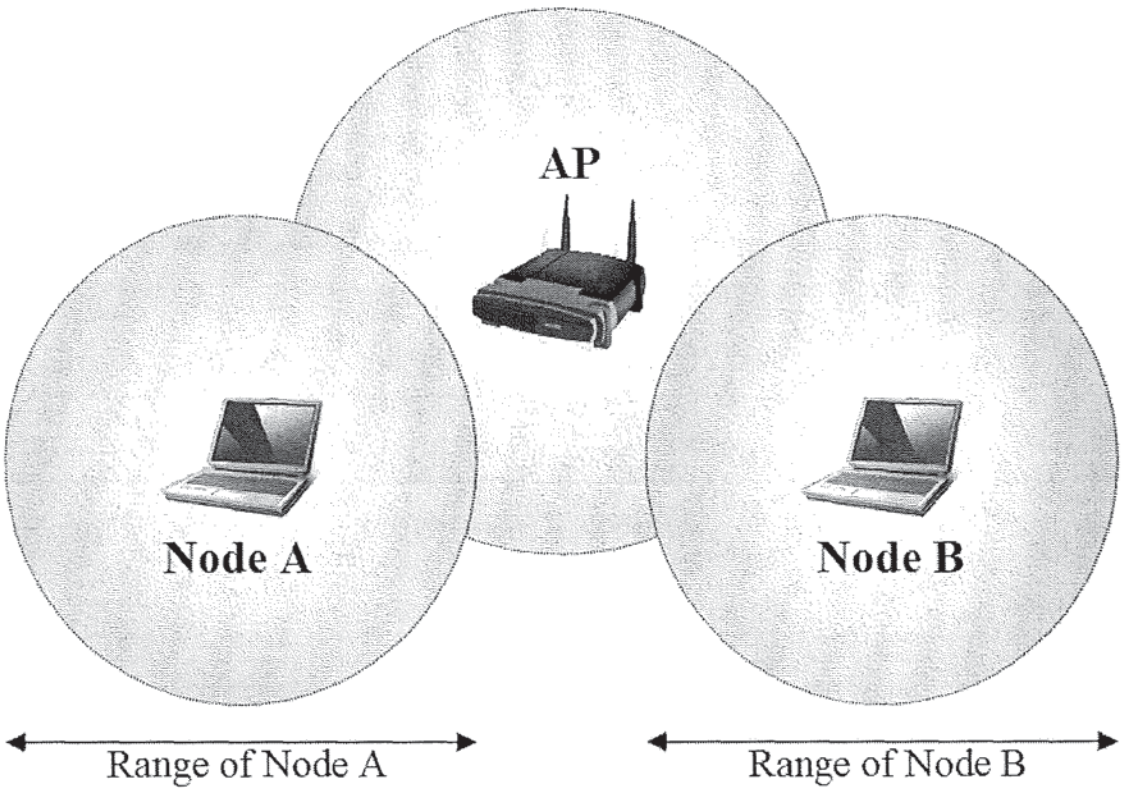


Figure 3.3: The hidden node problem in 802.11 WLANs

As illustrated by Figure 3.3, the hidden node problem occurs when an AP can communicate with both node A and node B, yet neither node A nor node B are in the range of each other due to limitations on transmission power with 802.11 devices. This creates a situation in which the AP could be receiving a transmission from node A

without node B sensing that node A is transmitting on the same channel. Node B, sensing no activity on the channel, might then also initiate a transmission on the channel. This leads to a 'jamming' of the AP's reception of node A's transmission, which is already taking place. The jamming is caused by frame collisions from both nodes, which will all be lost. This is known as the hidden node problem [143].

Another cause of frame collisions in 802.11 WLANs using the DCF CSMA/CA is if two or more devices that are randomly contending for access to the radio channel decrement their back-off timer to zero at the same time [144]. The simultaneous medium access and transmissions will lead to missing frames due to collisions. Each device must then generate a new value for their back-off timer from a range, which is increased each time when there is a failed attempt at acquiring access to the channel [23].

3.3 TCP Performance Issues over IEEE 802.11 WLANs

This section of the chapter presents a review of the recurring issues with TCP performance when it is used specifically over IEEE 802.11 WLANs.

3.3.1 Unnecessary TCP Retransmissions

One of the strengths of a reliable transport protocol like TCP is that it uses error detection coupled with a retransmission mechanism for recovering lost segments. Whilst these mechanisms work seamlessly in traditional wired networks, they have been shown to work inefficiently in wired-to-wireless environments [13]. TCP relies on network congestion signals to activate its various AIMD algorithms, which take responsibility for retransmitting missing segments and easing congestion in the path.

When TCP is used in conjunction with 802.11 WLANs many issues relating to its retransmission mechanisms have been reported [145] [146] [147].

Based on literature reviews in the area, and broadly categorising them, there are two key problems inherent to the characteristics of 802.11 WLANs which lead to unnecessary retransmissions of segments by TCP senders in the wired domain.

Firstly, the higher error rates of 802.11 WLAN channels lead to a greater number of TCP segment errors over the air. When an 802.11 frame containing a TCP segment is in error, the sending MAC uses its stop-and-wait ARQ in an attempt to recover the frame. While this is taking place, the TCP sender is still waiting for the arrival of an ACK to confirm a successful delivery. Unfortunately, the TCP sender's RTO timer expires due to a sudden increase in delay, and it is forced to retransmit the unacknowledged segment in concern. This situation is commonly referred to as a *spurious timeout* by a TCP sender [148], which has become a topic of much research in recent years [149]. Spurious timeouts at TCP senders are a common problem in wireless networks with high delay variability, and occur whenever there is a sudden and unexpected inflation of the RTT experienced by TCP, such that it exceeds the value of the RTO timer that has been calculated [150].

Meanwhile, the 802.11 MAC may have actually delivered the TCP segment successfully on one of its retransmission attempts, and has moved on to service the next TCP segment in its transmit queue. However, the MAC will now receive the retransmitted TCP segment it its transmit queue, which it will eventually service. Once delivered to the TCP receiver, it will be regarded as duplicate or expired data which the receiver has already received; hence the segment is simply discarded. Ultimately, this becomes a wasted effort on the part of TCP because the retransmission was unnecessary.

The spurious timeout problem described above can be further exacerbated if the retransmitted TCP segment is still not acknowledged in a timely manner. This can happen due to two reasons; i) if the 802.11 MAC ARQ is making many retransmission attempts of the retransmitted TCP segment due to persistent poor WLAN channel conditions, and ii) the retransmitted TCP segment is waiting at the back of a large transmit queue of the 802.11 MAC. Both of these situations will increase the delay experienced by the TCP sender, leading to a second successive RTO event due to the non-arrival of a TCP ACK in a timely manner. TCP doubles the value of its RTO timer each time it performs a retransmission, up to a maximum time of 64 seconds [49]. So for each successive RTO event, TCP waits for a longer period of time before responding to unacknowledged segments. This is known as the *exponential back-off*

problem, which will stall a TCP sender in the presence of actual losses, and has been reported in several studies [11] [106].

A second problem that can lead to unnecessary retransmissions by a TCP sender is when TCP ACKs are lost over wireless channels on their way back to the sending TCP in the wired domain. TCP ACKs are sent as 802.11 data frames by the MAC on receiving devices, so they go through the same transmission (and retransmission) procedures as any 802.11 data frame would do at the MAC of a sending device. The only difference with TCP ACKs is that if they are completely lost over the wireless channel due to the receiver's 802.11 MAC reaching its retry limit, then they are not retransmitted by the TCP layer on that receiving device. Hence they will not arrive at the sending TCP at all. This will lead to an expiration of the sender's RTO timer, forcing it to retransmit the unacknowledged data which has already been successfully received at the other end. The sending TCP therefore confuses an erroneous reverse channel loss for losses occurring in the forward channel, unnecessarily retransmitting a successful delivery. Note that DUPACKs can also go missing over the reverse WLAN channel as stated, and as such a TCP sender may not receive the three DUPACKs which are required to initiate a fast retransmission. Instead it incorrectly leads to an expiration of the RTO timer, with the sender initiating the slow start algorithm after the retransmission instead of the congestion avoidance algorithm [151]. Since DUPACKs imply that data is still arriving at the receiving end, initiating the slow start algorithm is an unnecessary degradation to the end-to-end throughput.

3.3.2 Suffering Throughputs for End-Users

One of the most popular measures of TCP performance by end-users is the end-to-end data throughput performance that is achieved at their end, and wireless end-users are no exception here. To clarify, TCP throughput is a measure of the amount of TCP data actually received by the TCP layer on the end-device per unit of time. It is a strong indicator of overall end-to-end performance as it is calculated in abstraction to the underlying wired-to-wireless conditions.

One of the key reasons for the success of 802.11 WLANs is that they claim to provide wireless end-users with high data transmission rates and bandwidths over the WLAN.

Consequently, 802.11 end-users over the years have become expectant of high throughput performance in support of the Internet applications which they use. However, TCP is the underlying dominant transport protocol that is used to meet such expectations by delivering application data from a server across the Internet, over an 802.11 radio channel, and eventually to the end-device.

Unfortunately, TCP finds it challenging to maintain high throughput levels when used over 802.11 WLANs due to the inherent nature of the radio channels used, and the problems discussed in the previous section. As a result it is unable to take full advantage of available capacity and claimed bandwidths of 802.11 channels [41].

The sufferings for TCP throughput arise from a combination of two key factors: i) constant retransmissions of segments due to variable delays and losses over the WLAN due to poor channel conditions, and ii) large delays over the WLAN associated with increased contention and medium access delays when there are multiple end-users being serviced by just one 802.11 AP in a WLAN BSS.

In the first situation, the limiting factor for TCP maximising its throughput is that of constant reductions in the size of the sender's *cwnd* each time a retransmission is carried out. In the case of RTO led retransmissions, the *cwnd* is reduced significantly to just one segment size because the slow start algorithm is initiated. In the case of retransmissions due to the reception of three DUPACKs, the *cwnd* is also reduced because the fast retransmit and fast recovery algorithms are initiated. Recall that the sender's *cwnd* strictly governs the amount of data that can be injected at once into the network per RTT. With continual reductions taking place during the transfer of data throughout a connection, the average throughput achievable is greatly limited by the sender's deficiencies. Consequently a TCP sender is unable to take full advantage of the high capacities available in 802.11 WLANs, i.e. a large *cwnd* size cannot be sustained at the sender.

In the second situation, large increases in the end-to-end delay for TCP senders cause a slow down in the rate of increase of the size of the *cwnd*, which translates into a less aggressive sending of new data to an end-user. The problem stems from the slowdown in the rate of arrival of ACKs at the sender from the receiver in the WLAN. This

happens when there are multiple 802.11 devices in a WLAN all contending for access to the radio medium. As discussed in a previous section, unsuccessful attempts in acquiring access to the wireless channel lead to an exponential increase in the random back-off interval for each failed transmission attempt for a device. The more devices there are, the more contention there is. This situation translates into an increase in the average medium access delay experienced by 802.11 data frames (containing TCP ACKs) in the transmit queue of a receiver trying to send TCP ACKs back to a TCP sender in the wired domain. Ultimately, the rate at which ACKs arrive at the TCP sender will decrease, thereby starving its *cwnd* of significant growth and affecting the optimal throughput on the forward channel.

3.3.3 Unfairness of TCP Flows in 802.11 WLANs

The 802.11 DCF MAC has been designed to operate as a fair protocol in the sense that all contending devices in a WLAN enjoy the same probability of gaining access to the radio medium in order to transmit their load. In fact, if there are N competing devices in the WLAN (including the AP), then each device has a $1/N$ probability of winning access to the medium. However, although this mode of operation ensures fair access to the medium at the MAC, it does not provide any provisions for ensuring fairness amongst TCP connections. The key issue is due to the fact that TCP data segments and ACKs are both transmitted as 802.11 data frames over the WLAN, which implies that all devices (including the AP) using TCP services to/from the Internet have a need for sole access to the radio medium to transmit data segments or ACKs.

There is no shortage of studies in this area, and generally, the consensus is that there appears to be a form of asymmetry between TCP traffic competing forward (downlink) and reverse (uplink) channels over 802.11 WLANs. A wide variety of issues relating to the unfair sharing of wireless capacity have been reported [152] [153] [154]. The issue is more prevalent when there are multiple devices all contending for channel access in a WLAN.

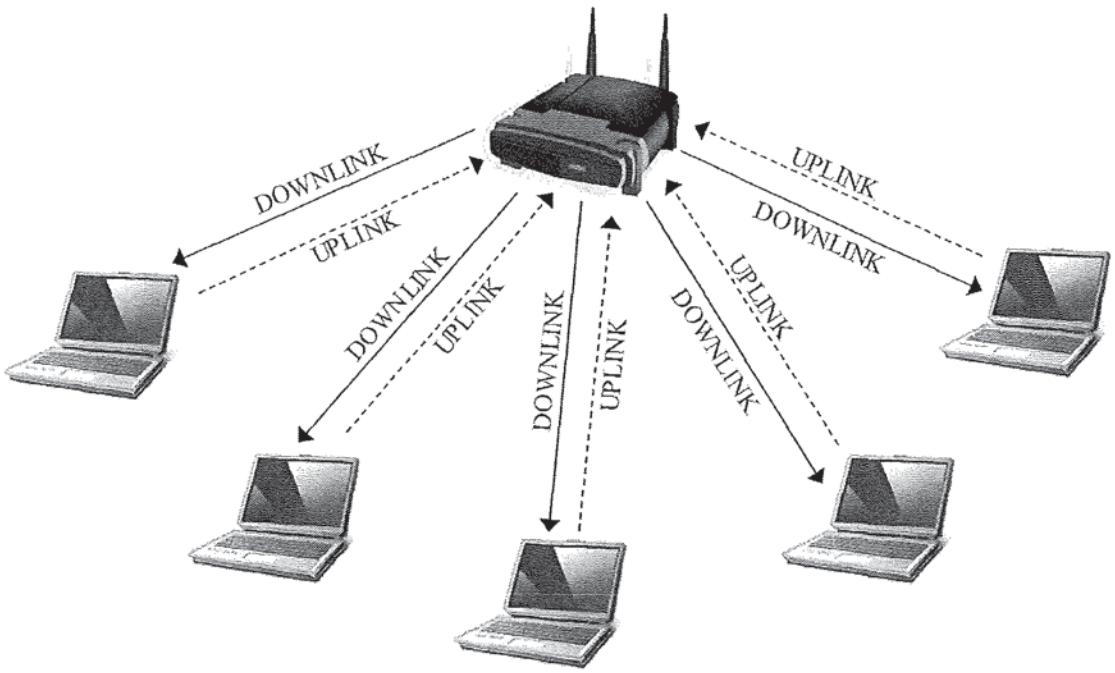


Figure 3.4: Concept of downlink (forward channel) and uplink (reverse channel) 802.11 flows

In essence, TCP unfairness results in significant degradations of performance for 802.11 end-users who use TCP applications [155]. There are two situations in which TCP unfairness can arise over an 802.11 WLAN operating in infrastructure mode: i) interactions between forward channel and reverse channel traffic of TCP connections, and ii) interactions occurring between only the uplink traffic flows of TCP connections. Figure 3.4 illustrates the concept of downlink and uplink traffic flows over an 802.11 WLAN.

In the primary case, downlink TCP connections are penalised by the presence of uplink connections. When there are multiple 802.11 devices in a WLAN each downloading TCP data from the Internet, it causes a buffering of incoming TCP data segments from multiple downlink requests at the MAC transmit queue inside the AP. As stated above, the AP does not have any privileges in terms of access to the radio channel in relation to the other N-1 end-user devices in the WLAN, and has only a $1/N$ chance of being able to transmit from its queue. In other words a single end-user device will have the same priority as that of the AP. The difference is that the AP has data in its queue for transmission to multiple end-users, whilst a single end-user device only needs to transmit towards the AP, i.e. uplink traffic gets a $(N\text{-}1)/N$ share of transmission opportunities. As a result, the downlink TCP connections can suffer from segment

losses due to buffer overflows inside the AP, because it is unable to service the queue at a quick enough rate. Such losses of TCP data segments will be fed back to the sending TCP in the wired domain, causing it to activate its congestion control procedures and retransmit lost segments. Note that this will also cause reductions in the sender's *cwnd* size, which will further starve throughput performance for 802.11 end-users performing download operations. Hence, unfairness exists for downlink TCP flows in certain scenarios [156].

In the secondary case, some of the problems of TCP unfairness arise from the interactions between multiple uplink TCP connections, i.e. from end-user devices to the AP. In this situation, the AP's transmit queue will be holding TCP ACKs which will have arrived from TCP receivers in the Internet. The AP therefore needs to deliver these TCP ACKs to the appropriate sender devices in the WLAN so that they can advance the sending of new data, and so forth. Unfortunately, a bottleneck of TCP ACKs can therefore occur at the AP when there are too many uplink connections occurring. This is because the AP again has only an equal ($1/N$) chance of gaining access to the medium, as explained above. As a result, buffer overflows can occur at the AP, and TCP ACKs will be dropped. Consequently, uplink TCP connections will be starved of sending performance due to a limited growth of the *cwnd* at each device, which are all dependent on the cumulative nature of returning TCP ACKs [51] [157].

In [158] it has been shown that there is also some unfairness between short-lived and long-lived TCP flows over an 802.11 WLAN. Here the short-lived TCP connections are more susceptible to the lossy nature of the radio channel as well as to the immediate unavailability of access to the medium that is needed in the initial stages of a TCP sender's *cwnd* growth phase, i.e. by the slow start algorithm.

3.3.4 Capture Effect on TCP Traffic

Although the 802.11 MAC is known to be a fair protocol, it has been discovered by several studies that there exists some form of unfairness which favours 802.11 devices with higher received signal strengths (i.e. that are closer to an AP). This phenomenon is caused by the so-called *capture effect* within contention-based WLANs [40] [159]. It arises when given two received signals arriving from different devices; a receiver will

consider the significantly stronger reception as the dominant signal, and weaker signal as interference noise. The receiver will therefore decode the stronger signal from the device that is closer, and the other device fails its transmission attempt due to detecting a collision [160].

In [40], it was discovered that the channel capture effect can lead to degradations from certain end-users when there are several TCP connections active over an 802.11 WLAN. In this study, the binary exponential back-off mechanism of the 802.11 DCF has been shown to favour the most active TCP connection. Hence when there are multiple end-users accessing TCP services from the Internet, those connections starting early or the most heavily loaded may have a higher probability of capturing the radio medium for transmissions due to the way in which the 802.11 DCF allocates time slots to transceivers following a recent transmission over the radio channel.

In [161], the authors demonstrated through experimental work there are serious unfairness issues related to the capture effect in the distribution of 802.11 WLAN channel bandwidth amongst multiple TCP end-users. It was caused by the differing received signal strengths of devices based on their physical location in proximity to the AP. The study clearly shows that when two end-users are performing TCP downloads simultaneously from a wired domain the performance of the device with the lower signal quality was disproportionately affected by the device with the higher signal quality.

3.3.5 Self-Collisions of TCP Traffic

Due to the bidirectional nature of TCP connections, a successful transmission of a single TCP data segment over an 802.11 WLAN requires at least four separate transmissions over the medium, i.e. the AP and an 802.11 device perform two transmissions each, as illustrated in Figure 3.5. This can be explained by the simple fact that both TCP and the 802.11 MAC offer a reliable transmission service using acknowledgements. To illustrate, a TCP data segment is sent by the AP as an 802.11 data frame over the WLAN, which is acknowledged by an 802.11 ACK from the receiving device. Once received, the TCP data segment is passed up to the TCP layer, and the receiver too acknowledges the data by transmitting a TCP ACK back to the

sending TCP via the AP. This TCP ACK is transmitted as a standard 802.11 data frame by the device, which the AP MAC must acknowledge using an 802.11 ACK [141]. Hence there are two attempts at contention of the wireless medium per a single TCP data transmission, once for the TCP data segment, and once for the TCP ACK segment.

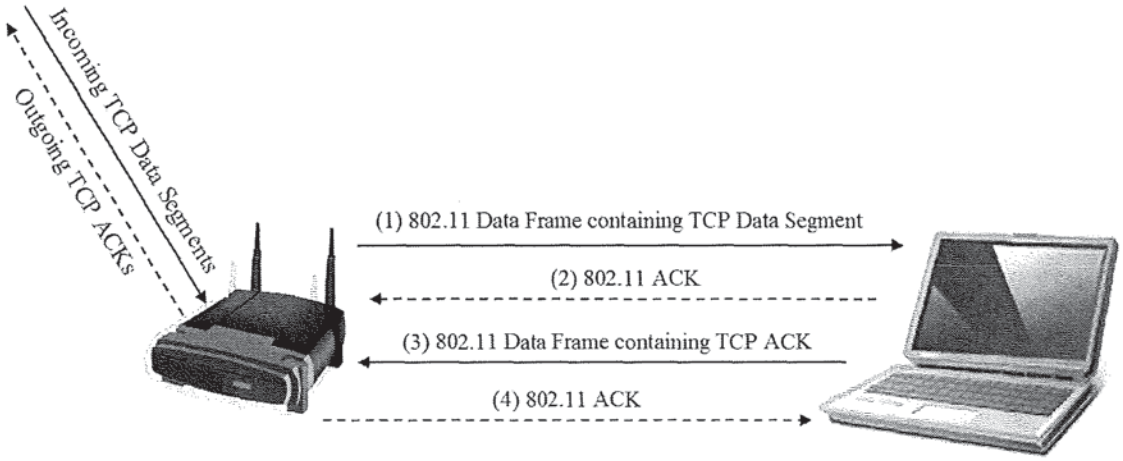


Figure 3.5: The four exchanges between the AP and end-device per TCP data segment

It is the two-way flows of TCP data segments and TCP ACKs described above between a pair of 802.11 devices that lead to the problem of *self-collisions* of TCP traffic over the air. TCP self-collisions occur when TCP data frames on the forward channel collide with TCP ACK frames on the reverse channel, and the problem is exacerbated when there are multiple TCP end-users in an 802.11 WLAN. The contentions between transmissions of TCP data segments and TCP ACKs result in collisions of TCP traffic, which lead to losses of data and ACKs for TCP connections. In [38] and [160] it has been shown that the collision probability of two-way traffic increases proportionally to the number of devices in an 802.11 WLAN.

3.4 TCP Enhancements for Wired-to-Wireless Paths

To clarify, the focus of this thesis is on the evaluation of sender-side TCP performance over wired-to-wireless paths, where the wireless path of the journey is typically an IEEE 802.11 WLAN. This thesis therefore favours those TCP enhancements that can be applied easily to an existing infrastructure of the Internet, i.e. at TCP senders. Here the TCP senders are typically servers in the Internet sending data to wireless end-users within home and office environments.

Now that the performance issues from the wider research community relating to sender-side TCP over wired-to-wireless paths (including over 802.11 WLANs) have been reviewed and brought to the reader's attention, this section aims to review some of the key approaches taken by researchers in recent years to enhancing TCP's end-to-end performance over such environments.

Specifically, the section focuses on those proposals which adhere to TCP's original end-to-end semantics, keeping all modifications/enhancements at the TCP-layer on the sender-side. Such solutions are the easiest to deploy in the Internet across servers, and can give performance improvements to wireless end-users globally without the need for modifications to end-devices. Such a technique therefore offers scope for excellent scalability across the Internet, which is a mandatory design feature if any TCP proposal is to be adopted by an increasingly heterogeneous end-user base.

3.4.1 Approaches for Enhancing Wired-to-Wireless TCP Performance

Before focussing specifically on the target category of proposals, a brief review of the different categories into which TCP solutions for wired-to-wireless environments generally fall is presented.

3.4.1.1 End-to-End Solutions

A reliable transport protocol such as TCP must provide true end-to-end semantics to applications, which implies that TCP ACKs must absolutely certify that the sent data has reached the receiver successfully and with the integrity of the data maintained throughout the journey to the receiver [3]. Figure 3.6 illustrates the concept of an end-to-end solution in a wired-to-wireless environment.

A true end-to-end TCP scheme for wired-to-wireless paths does not require the involvement of any intermediaries for managing its flow and congestion control. Intermediaries include any devices in the path between the sender and receiver, such as the AP acting as a gateway into the wireless path [45].

Furthermore, end-to-end schemes do not require modifications to intermediaries either; all changes to the TCP code are restricted to the TCP layer at either the receiver side, or the sender side. This allows for ease of integration into an existing infrastructure.

Generally speaking, TCP enhancements in this category can be regarded as 'fixes' to its original intended behaviour, such as ACK-clocking, maintaining an RTO timer, altering the *cwnd* size in light of congestion, and so forth. The idea of an end-to-end solution is therefore to ensure that TCP (and its congestion control features) behaves as it should under all circumstances, but at the same time making it more robust in new heterogeneous environments that may not have been envisaged upon its original appearance.

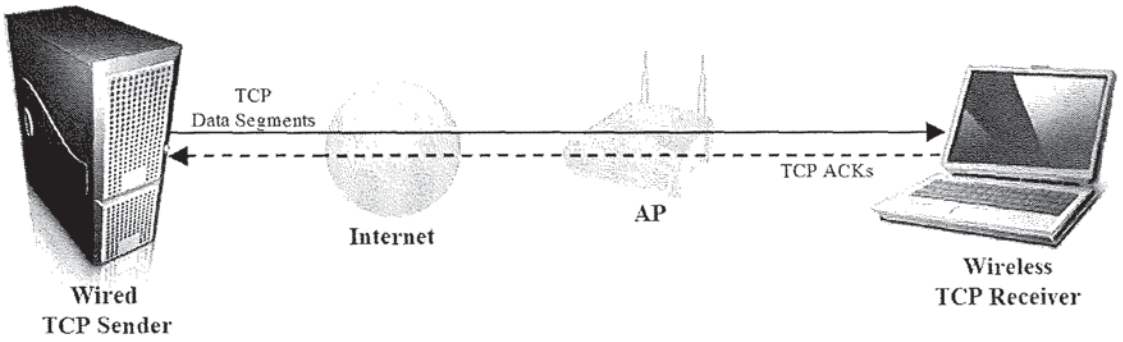


Figure 3.6: Concept of an end-to-end solution for TCP over wired-to-wireless paths

TCP NewReno, the SACK option implementation, and the delayed-ACKs mechanism are all fine examples of end-to-end fixes that were made to the original TCP to enhance performance in a changing Internet demanding consistently higher throughputs; here the modifications were made to the TCP code at the end-points only, and with relative success.

The advantages of end-to-end schemes are fairly obvious from the discussions above, however they do also have some disadvantages [162]. Most end-to-end solutions do not have any knowledge or sense of the conditions of the physical layer communication path, and because of this end-to-end solutions are usually based on mechanisms that can intelligently discriminate between the causes of TCP segment loss in an end-to-end path; either a congestion related loss or a segment loss over the wireless path. The same is also true for any unexpected delays that a TCP may

encounter; is the delay due to congestion in the fixed network or is it due to the characteristics of the wireless path. A further disadvantage is that an end-to-end TCP solution may not be precise or quick enough at dealing with problems over the wireless path, as it based on assumptions and measurements. The additional processing complexities can also increase overheads at TCP senders in the Internet.

3.4.1.2 Connection Splitting

It is possible to enhance TCP performance without making any changes to TCP implementations at end-points. This can be carried out by placing an intermediary device in between the end-to-end flow of a TCP connection that transparently alters the flow of traffic, leading TCP to believe that conditions in the network are better than what they may actually be [3]. The intermediary is technically referred to as a *performance enhancing proxy* (PEP), and there are official guidelines on its proper usage within networks for enhancing performance [163]. A PEP can also be implemented at any layer within an intermediary device. The idea is to 'fool' a TCP sender or receiver by forcing it to behave desirably in order to bring about more stable behaviour and consequently better performance. Figure 3.7 illustrates the concept of splitting a TCP connection between the wired and wireless domains.

In wired-to-wireless environments, a PEP is typically implemented at the interchange between the wired domain and the wireless domain. Usually, this would be at the gateway to a wireless network, such as at the AP for 802.11 WLANs. Hence, solutions that are based on this method are known as *connection splitting* schemes, because an end-to-end TCP connection is split into two separate connections, one running over the wired path and one running over the wireless path [13].

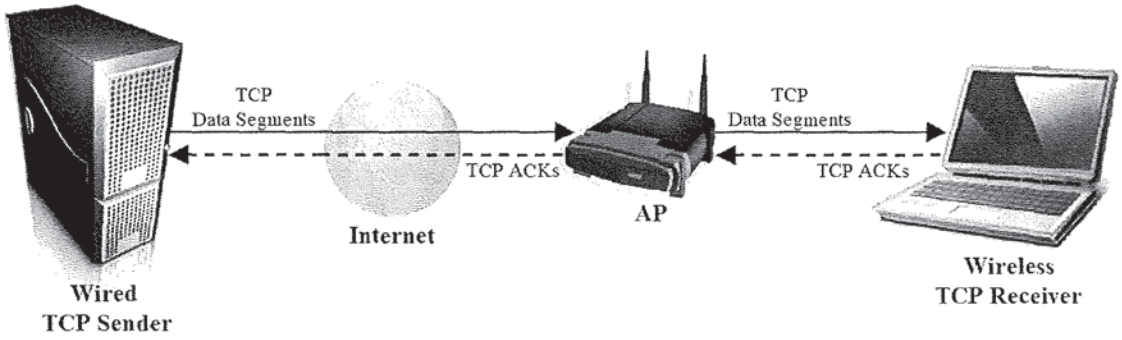


Figure 3.7: Concept of splitting a TCP connection at the gateway to the wireless domain

The AP maintains a TCP connection with a server in the wired domain using a regular version of TCP supported by the sender, while at the same time it maintains a TCP connection with an end-user in the wireless domain using an enhanced transport protocol that is optimised for wireless channels. Both end-points therefore communicate directly with the AP without prior knowledge of the other end of the link [30].

In the majority of implementations, the PEP has permission to acknowledge TCP segments as soon as it receives them. Unfortunately, this can lead to a violation of TCP's end-to-end semantics because TCP ACKs can arrive at the sender before the segments are actually delivered to the receiver [30]. Another feature of connection splitting that can also violate TCP's end-to-end semantics is when segments pass through the AP, the PEP usually has to access and reveal certain information contained within the passing segments before they reach their destination. The advantages of connection splitting TCP schemes is that generally they can respond to wireless related events very quickly, and no modifications are required to TCP data senders in the wired domain.

Indirect-TCP (I-TCP)

A well known connection splitting solution for TCP over wired-to-wireless environments is *Indirect-TCP* (I-TCP) [164], which has been the classical solution for many years now for last-hop wireless networks. I-TCP utilises a PEP at the AP to establish two separate TCP connections, as mentioned above. The AP therefore manages a *cwnd* independently for each connection. In I-TCP, only the implementation of TCP used over the wireless path is modified, with added support for mobility and disconnections of wireless devices. With regards to performance, I-TCP yields only small improvements over a regular end-to-end TCP connection for wireless channel BERs up to $2x10^{-6}$, after which the throughput performance is on comparable terms. A drawback of I-TCP is that it cannot deal with failures or rebooting of the AP, which lead to TCP connections being terminated and data being lost.

Selective Repeat Protocol (SRP)

Selective Repeat Protocol (SRP) [26] is another example of connection splitting in which a TCP connection is split at the AP. Connections between fixed wired hosts and the AP use a regular TCP implementation; however an enhanced transport layer protocol is used between the AP and the wireless devices it supports. The enhanced transport protocol is able to selectively repeat transmissions of lost segments over the wireless path.

Unlike using a standard end-to-end TCP connection, SRP has the rather significant advantage of being able to recover more than one segment per single RTT. It uses the SACK option at the AP to specify non-contiguous blocks of missing segments. The AP, upon receiving SACK information, is able to locally retransmit a series of missing segments, which could have been lost from a single *cwnd* due to errors on the wireless channel. Another advantage of using SRP can be realised when data is sent upstream from a wireless device to the wired TCP sender. The MTU of a wireless link is usually much smaller than a wired network. Using regular TCP for the entire connection path (i.e. over the wireless channel too) would mean the same MTU size must be used over the wired path too. Using SRP on the other hand, smaller segments are reassembled at the AP to take the advantage of larger MTU sizes from the wired network.

However, a significant drawback of the SRP solution is that the AP has to constantly maintain hard-state information and then transfer the state to another AP when handovers occurs. The TCP sender in the wired domain may also receive ACKs before data segments are actually delivered to the wireless receiver, thus violating end-to-end principles.

Overall, there are some significant disadvantages of using TCP connection splitting to enhance performance over a wired-to-wireless path, which are mainly due to violation of the end-to-end argument for TCP. Other issues also exist that are mainly to do with the AP. APs would need to support a PEP implementation for which, in today's market dominated by IEEE 802.11 APs, there is yet to be any signs of progress with connection splitting mechanisms in the mass market. Secondly, if an AP is to support a PEP implementation, then it would also need advanced buffering capabilities, which can prove costly. Thirdly, a TCP segment arriving from the fixed domain would need

to traverse two TCP stacks at an AP, as opposed to zero traversals when using a standard approach. Many attempts have been made in recent years to enhance the functionality of PEPs in an attempt to make them more desirable for connection splitting approaches [165]. Generally speaking, connection splitting solutions often suffer from high software overheads due to duplication of protocol stacks at the AP.

3.4.1.3 Link-Layer (LL) Schemes

The major problems for reliable transport protocols like TCP arise because of the unpredictable nature of the wireless medium. Therefore, if the problem could be tackled at its root cause, shielding TCP from the lossy link, then any such solutions would appear ideal. Since the *data-link layer* protocol of an AP is running on top of the PHY, it has immediate knowledge of error conditions over the medium. It therefore has the potential to react faster than higher-layer protocols, with the hope of shielding any problems from them too. Such solutions are referred to as link-layer (LL) schemes. Although not strictly an enhancement to TCP directly, LL solutions do have their place in the field of research for TCP performance over wired-to-wireless networks [26]. They hide the characteristics of the wireless link from the transport layer by attempting to solve the problem at the link layer. In fact, the IEEE 802.11 MAC sub-layer is a LL protocol that does precisely this.

A LL protocol at an AP also has more control over the PHY; hence it can alleviate inefficiencies of the wireless medium, providing a TCP sender in the Internet with a reliable end-to-end communication path similar to that of a purely wired path. Such a technique does not require the need to make changes to existing TCP implementations on end-hosts, which can be seen as advantageous because the end-to-end semantics of TCP are preserved [13].

Unlike the transport layer, which has TCP as the defacto protocol, the link-layer has no set protocol. Also, attempting to make the wireless channel appear as a wired channel is not as straight-forward as it may seem. An LL protocol's main function is to ensure reliable delivery of packets over a wireless network, much in the same way as TCP's core end-to-end functionality. Hence, LL schemes typically employ a variety of local error control techniques to achieve this [166], including:

- *stop-and-wait ARQ*
- *go-back-N ARQ*
- *selective-repeat (SR) ARQ*
- *forward error correction (FEC)*

Consequently, LL solutions for enhancing TCP performance over wired-to-wireless paths are more likely to be able to respond quickly to errors in the wireless path.

Past studies have shown that ARQ mechanisms of LL protocols can work well for relatively low error rates; high error rates, however, can lead to a greater number of local retransmissions, and can sometimes lead to a connection 'black-out' for TCP senders in the wired domain [167]. Alternatively, the use of FEC can detect and reverse a specific number of erroneous bits per wireless packet, but the bandwidth penalty can be quite significant in terms of the actual data throughput for end-users. Hence, the use of FEC is not greatly suited to channels with limited bandwidth, which is usually the case for wireless networks. FEC usage also increases the processing delay per packet [167].

Moreover, it is a known fact that LL retransmission timers should expire earlier than TCP's RTO timer in order to avoid repeated retransmission efforts from TCP senders in the Internet. In other words, LL protocols should try to avoid triggering TCP retransmissions unless it is absolutely necessary, for example by not delivering wireless packets out-of-order or by suppressing redundant TCP DUPACKs, and so forth. Such duplicated efforts can lead to worse overall performance over the wireless channel due to wasted resources and triggering of TCP's congestion control mechanisms unnecessarily [26]. In summary, LL schemes need to consider two critical factors: i) determining the LL retransmission timeout period and setting the maximum number of retransmission attempts per loss, and ii) considering the interaction between itself and the impacts it may have on TCP's retransmission behaviour. Based on these criteria, LL solutions generally fall into one of two categories respectively; i) *TCP-unaware*, and ii) *TCP-aware*.

Snoop Protocol

The most renowned LL proposal in the TCP-aware category is the *Snoop* protocol [168] [169]. Snoop's design requires no changes to the wired infrastructure of the Internet, since it is implemented locally over the wireless path. Snoop employs an *agent* that resides at the AP, and requires wireless end-devices to run the Snoop client protocol. The Snoop agent runs as an additional sub-layer inside the AP above a LL protocol. The agent's function is to inspect each and every TCP segment and ACK that passes through the AP, in either direction. Those that are destined for wireless end-hosts are stored in a local *cache* of unacknowledged TCP data. If in case the Snoop agent encounters a DUPACK on its way to a wired TCP sender, the agent suppresses the DUPACK instead of forwarding it on to the sender. It then retransmits the missing data segments immediately from its local cache.

As an additional feature, the Snoop agent also maintains its own retransmission timer, so that it can perform local retransmissions without TCP having any awareness of it. Of course, the agent's timer must be less coarse than TCP's own timer in order for a local retransmission to take place (plus its associated LL acknowledgement) within the time frame of TCP's timer to avoid a RTO event at the sender.

Snoop's strengths lie in the fact that it takes advantage of LL protocols' ability to respond quickly to wireless channel errors, while simultaneously it is able to use this information to keep TCP in a stable state over the existing connection. It also maintains TCP's end-to-end semantics.

It has been shown that the use of Snoop can yield better performance than a regular TCP connection running end-to-end over the same path [26]. One of the features of Snoop that stands out is its ability to respond to segment losses faster than TCP.

A drawback of using Snoop is the fact that TCP retransmissions still do occur in coexistence with local LL retransmissions, mainly due to timeouts at the sender [13]. Certain studies have also shown that Snoop is most profitable when nearly all data flows are from the wired domain towards the wireless domain [30]; this is due to the way in which the Snoop agent performs local error recovery. Another drawback is that Snoop was designed for TCP connections with small RTTs; hence, studies have shown that a Snoop protocol does not necessarily yield enhanced TCP performance in wired-

to-wireless last-hop scenarios that typically have larger RTTs due to the Internet path [170]. This is a significant flaw since the majority of TCP connections today are between a server in the Internet and a wireless receiver at the edges of the Internet, where end-to-end RTTs are typically larger.

Delayed Duplicate Acknowledgements (DDA)

A LL proposal that falls clearly into the category of TCP-unaware solutions is the *delayed duplicate acknowledgements* (DDA) scheme [171]. The term 'TCP-unaware' refers to the fact that DDA operates independently to TCP and is not aware of what is happening at the TCP layer at either endpoint. In summary, DDA attempts to emulate the workings of the Snoop protocol, and has two main goals; i) to retransmit TCP segments using a LL protocol, and ii) to prevent the TCP sender from retransmitting segments that have already been supplied by the LL protocol at the AP.

The DDA scheme requires that each TCP data segment is encapsulated within a LL frame, and each TCP ACK is encapsulated within a LL-ACK frame. DDA also uses a different numbering system for keeping track of locally transmitted frames. When an AP receives a LL-DUPACK frame (containing a TCP DUPACK), it assumes there has been a loss. The AP then delays the forwarding of the DUPACK to the TCP sender by a fixed amount of time (using a local timer). At the same time it retransmits the lost segment locally. In the meantime if more LL-DUPACKs arrive then they are also delayed. If within the fixed time a LL-ACK arrives, indicating the successful reception of a retransmitted frame, the delayed LL-DUPACKs at the AP are discarded. If there is an expiry of the local DDA timer, then all TCP DUPACKs are released and forwarded to the TCP sender in the wired domain, which will force it to activate the fast retransmit algorithms (if there are three or more).

The main advantage of the DDA scheme is that the LL protocol at the AP does not need to be aware of the workings of TCP. Consequently, TCP's end-to-end semantics are maintained, with no modifications necessary to wired hosts. The authors of [171] suggest that their scheme works well in cases where the delay over the wireless path is small in comparison to the RTT of the wired path, i.e. it does not perform well on slow wireless links. This is mainly due to the expiry of TCP's retransmission timer before the DDA local timer expires, causing unnecessary duplicate retransmissions. Another

potential drawback is that the DDA scheme does not attempt to deliver segments in-order to TCP because of the independent local numbering system it uses.

In contrary to some of the obvious benefits of deploying a LL scheme for enhancing end-to-end TCP performance over wired-to-wireless paths, LL solutions do have their disadvantages. Firstly, in very poor wireless channel conditions, LL packets can suffer from heavy losses and reordering that will cause large fluctuations of the RTT measured by TCP in the wired domain. This can lead to spurious timeouts and many redundant retransmissions by TCP, as well as significant reductions in its sending rate because it still regards all problems as signals of network congestion. Secondly, LL schemes are unable to deal with frequent disconnections of a wireless device that a TCP sender in the Internet is communicating with.

3.4.2 Sender-Side End-to-End Approaches

To date, there have been many attempts and proposals in the quest for the perfect end-to-end TCP implementation for wired-to-wireless paths. In essence, a perfect end-to-end TCP should be able to accurately differentiate between congestion-related losses and wireless-related losses (otherwise referred to as random losses), and then act accordingly to sustain a stable behaviour within itself and providing adequate throughputs for all parties concerned. Similarly, it should also be able to discriminate between the causes of unexpected delays; whether they are due to network congestion, or whether they are associated with the wireless path.

Although the focus of the thesis is on those enhancements that are implemented on the sending (fixed) side of a TCP connection, many end-to-end solutions have also been aimed at the receiving (wireless) side of connections, i.e. modifications made to the TCP implementation on wireless devices. The most prominent receiver-side TCP enhancement proposals in the area include *Freeze-TCP* [172], *TCP-Peach* [173], *WTCP* [170], *TCP-Casablanca* [6], *TCP-Real* [174], *Explicit Transport Error Notification* (*ETEN*) [175], *E2E-ELN* [129], and *TCP Handoff* (*TCP-HO*) [176].

Placing the focus now on the sender-side implementations of an enhanced TCP, there is an array of techniques and approaches taken by researchers over the years to

differentiate between the causes of loss and delays in a wired-to-wireless path. The basic idea is to avoid triggering a TCP sender's congestion control algorithms when a loss or delay is detected over the wireless path, i.e. random losses and delays, thereby maintaining a high sending rate for wireless end-users.

A sending TCP-layer always has ready access to information such as its *cwnd* evolution and size, the receiver's *rwnd* size, the inter-arrival times between ACKs, the current RTT and RTO values, and deviations in the RTT, to name the most commonly used [177]. Therefore, sender-side enhancements typically resort to using a combination of these metrics to intuitively gauge the most likely cause of losses or delays at any one time, and then respond accordingly to maintain sending rates. However, the greatest challenge with this approach is for TCP to use its congestion control algorithms appropriately by being able to probe for available bandwidth in such heterogeneous paths [11].

Based on a thorough review of literature in the area, sender-side end-to-end TCP enhancements can be grouped into one of several categories of solutions.

3.4.2.1 Reactive Solutions

A *reactive* TCP AIMD algorithm deals with and attempts to resolve an incident of network congestion when specific thresholds have been reached. It reacts by making necessary adjustments in order to rectify the situation, by reducing its *cwnd* size and sending rate for example.

TCP Reno

The legacy TCP Reno [99] is in fact a reactive AIMD algorithm; it adjusts its *cwnd* in response to the timing and feedback of incoming ACKs and DUPACKs. It also continues to probe for more bandwidth in a communication path until the onset of congestion is inevitable, at which point it reacts again by cutting back its *cwnd*, and so forth. However, when the indicators of congestion (i.e. segment losses/delays) are random and not due to real congestion, then a slower sending rate is unnecessary and the incorrect response by TCP.

SACK Option

The TCP SACK option [112] is an early example of a reactive enhancement option to TCP Reno, which was proposed as a means for a sender to detect and recover multiple segment losses from a single *cwnd* of data in flight in a single RTT [110]. The SACK option is now implemented widely in the Internet and enabled by default into the TCP stacks of all major end-user operating systems [178]. Unfortunately, SACK still remains a reactive mechanism, and therefore has its limitations [179] [180]. Each SACK block needs 8 bytes of header space to convey its information to the sender; hence the number of blocks that can be conveyed per TCP ACK is limited to just three when the timestamps option [109] is also enabled. This restriction can lead to unnecessary retransmissions of successfully received segments under certain loss conditions, as illustrated in [179]. Hence many researchers argued if there was any advantage in using the SACK option in lossy conditions, such as over wireless channels. In [181] a more conservative loss recovery algorithm based on the usage of the SACK option is proposed.

TCP NewReno and Variants

TCP NewReno is another early example of a reactive sender-side enhancement to the legacy TCP, for use where the SACK option could not be supported [182]. The key enhancement with NewReno is that (unlike Reno without SACK) it is able to distinguish between a *partial*-ACK and a *full*-ACK. A full-ACK acknowledges all segments that were outstanding at the start of the fast recovery phase, while a partial-ACK acknowledges some but not all outstanding data. Unlike Reno, where a partial-ACK will terminate fast recovery, a NewReno sender will retransmit the next in-sequence segment based on the partial-ACK, reducing its *cwnd* by one less than the number of segments acknowledged by the partial-ACK. This window reduction, referred to as a *partial window deflation*, allows the sender to transmit new segments in subsequent RTTs of the fast recovery phase. On receiving a full-ACK, the sender sets *cwnd* to *ssthresh*, terminates fast recovery, and resumes congestion avoidance [182].

A problem occurs with NewReno when there is a reordering of segments by more than three sequence numbers [113]. When this happens, a NewReno sender mistakenly invokes the fast recovery algorithm. When the reordered segment is eventually

delivered, segment sequence numbers will progress and from there until the end of fast recovery, every new ACK produces a duplicate and unnecessary retransmission.

In [183] and [184] the authors conclude that NewReno works well when there are multiple segment losses in a single *cwnd* and that retransmissions are usually successful. But when the retransmissions are also lost due to an erroneous link in the path, they cannot be recovered quickly enough by NewReno, and they also cannot be fast-retransmitted because of the lack of DUPACKs. Therefore successive RTO events are inevitable.

Today there are two variants of TCP NewReno implementations in circulation across the Internet that can recover from multiple segment losses: the *slow-but-steady* variant and the *impatient* variant [113]. However, both variants suffer from slow recovery times when many segments are lost. In fact, the recovery time of the slow-but-steady variant increases linearly in proportion to the number of lost segments, and the impatient variant, although it performs better than the slow-but-steady variant in the case of multiple losses, it wastes time waiting for a RTO timeout before entering the slow start phase [185]. In [185] the authors therefore propose the *instantaneous* variant of NewReno as an alternative sender-side enhancement.

DSACK Extension

The *duplicate selective acknowledgment* (DSACK) extension [186] to the TCP SACK option has recently been proposed to make TCP more robust to segment reordering problems that commonly occur in lossy conditions. It specifies the usage of the existing SACK option header field to report which sequence numbers are generating the ACKs. This allows a TCP sender to infer the actual order of segment arrivals at the receiver, and when it has unnecessarily retransmitted a segment, i.e. due to a spurious retransmission. The information contained in the DSACK field can assist TCP with adjusting its sending behaviour to improve end-to-end performance.

The reception of DUPACKs can be an indication to the sender of either segment reordering or even losses. The ability to discriminate between these two cases impacts TCP behaviour and performance considerably. If there is persistent reordering in a heterogeneous network path, over a wireless path for example, then it will generate

many DUPACKs. If the sender interprets the DUPACKs as losses, then upon receiving *dupthresh* number of DUPACKs (where *dupthresh* is normally set to three as defined in [99]), it will invoke the fast retransmit algorithm. If the fast retransmit algorithm is activated frequently to resend segments that may not have been lost, it is very wasteful of network bandwidth and keeps the average *cwnd* size unnecessarily small [90].

There have been several attempts in recent years by researchers capitalising the DSACK extension to better gauge path conditions at the sender by detecting and avoiding false invocations of the fast retransmit algorithm. The general approach taken is to improve TCP performance by adaptively modifying the value of *dupthresh* throughout a connection [146] [187] [188]. However, the proposal in [187] has been criticised for not being able to adapt the value of *dupthresh* accurately enough in real-world conditions leading to RTO events when multiple segments are lost from the same *cwnd*, and with proof of concept coming only from simulation work. In [188] the authors propose *RR-TCP*, an algorithm based on DSACKs allowing a sender to recover from false retransmissions, and avoid future unnecessary retransmissions by dynamically adjusting the *dupthresh* value. Although RR-TCP is able to improve TCP performance in the presence of severe reordering (as shown by simulation work), it suffers from being too computationally expensive due to storage overheads and requirements at the TCP-layer.

The approach taken in [146] builds on previous work related to DSACK usage by proposing a sender-side algorithm that uses an *exponentially weighted moving average* (EWMA) coupled with the mean deviation of the length of the reordering event reported by a TCP receiver in a DSACK to estimate the value of *dupthresh*. It also uses an adaptive upper-bound on *dupthresh* to avoid RTO events. Although the approach in [146] has been shown to slightly enhance TCP performance over paths with lossy links, the authors verified their algorithm using simulation studies only. It is therefore questionable whether or not the algorithm would be stable in real-world conditions, where the RTT, segment reordering, and losses are unpredictable.

k-SACK

The *k-SACK* sender-side enhancement has been proposed in [189], which uses information conveyed via the regular SACK option to improve TCP performance over

lossy links with random losses, with references made to wireless links. Its key functionality is to not consider all segment losses as an indication of network congestion. The k-SACK algorithm uses the notion of a *loss window* and the parameter *k*, which is the threshold for determining whether losses are congestive or random. Then for each *cwnd* of sent segments, if out of the *loss window* number of most recently transmitted, *k* or more segments are lost, a k-SACK sender assumes network congestion, otherwise the losses are assumed to be randomly occurring due to a lossy link in the path. Note that the values of the parameters *loss window* and *k* are predetermined in an appropriate manner. However, a large value of *k* increases the time it takes a TCP sender to detect and respond to congestive losses. Secondly, the authors state that high values of *k* are also likely to have an adverse effect on competing TCP connections across the Internet, and hence the value of *k* is a subject for future work.

The focus of k-SACK is to prevent drastic reductions of a sender's *cwnd* size for random losses, but only for congestive losses. The algorithm modifies the fast recovery algorithm of the legacy TCP to operate in accordance with the value of *k*, where the *cwnd* is frozen when the number of losses is less than *k*. Although simulation results highlight the effectiveness of TCP with k-SACK over the regular SACK option over a range of random loss rates, the authors used only a basic dumbbell topology. Errors are artificially induced over the bottleneck wired link, following an *independent and identical distribution* (i.i.d). Unfortunately, burst errors are not considered in the experiments, and neither is a wireless hop that utilises a contention based channel access mechanism, as well as a local error recovery mechanism. Therefore it is difficult to predict how k-SACK would perform in real-world conditions across the Internet with a last-hop 802.11 WLAN, where congestive losses along the wired path and random errors in the wireless path can coexist, and where there could be interplay between the TCP and 802.11 MAC error recovery mechanisms.

SACK+

A fairly recent reactive sender-side modification to TCP is *SACK+* [190], which was proposed as a supplement algorithm for those senders already using the popular SACK option. The motivation behind SACK+ is that it aims to alleviate the deficiencies of the standard SACK option, where lost retransmissions are undetectable until there is an

expiry of the RTO timer at the sender [191]. The SACK+ algorithm claims to be able to detect retransmitted segments that are also lost. The authors use a stochastic modelling approach to evaluate the performance of SACK+, supported by numerical results from simulations to highlight its improvements over the regular SACK option. However, many assumptions are made with the stochastic model, with the most significant being the omission of an ACK loss model on the reverse path of TCP connections. Finally, the authors go on to suggest that the improvements offered by SACK+ in the real-world may be insignificant because segment retransmission losses are infrequent events for TCP in the Internet. For those TCP implementations that are unable to support the standard SACK option, the authors have proposed the *duplicate acknowledgment counting* (DAC) algorithm as a sender-side TCP enhancement [192]. The DAC algorithm relies on the arrival of DUPACKs instead of SACK information to detect lost retransmissions, but as with SACK+, its merits are based only on quantitative evaluations.

3.4.2.2 Proactive Solutions

A *proactive* TCP algorithm uses feedback mechanisms from within the network to direct a sender towards making continual adjustments that can prevent the onset of congestion, and any other related issues. It is the more preferred type of solution for enhancing TCP performance over wired-to-wireless environments [11].

TCP Santa Cruz

One of the first proactive sender-side TCP enhancement proposals targeted at wired-to-wireless (or heterogeneous) paths was *TCP Santa Cruz* [167]. The implementation of Santa Cruz introduced new congestion control and error recovery mechanisms to deal specifically with lossy links and dynamically changing path delays. Its originality is due to the fact that it did not use the RTT of segments in any way for congestion control. Instead it uses the notion of the *relative delay* of segments. The *relative delay* is the increase/decrease in delay that segments experience with respect to each other as they propagate through the network. These measurements are the basis of the congestion control algorithm in Santa Cruz. A TCP sender calculates the *relative delay* from a timestamp contained in every ACK that specifies the arrival time of the packet at the receiver. From the *relative delay* measurement the sender can determine whether

congestion is increasing or decreasing in either the forward or reverse path of the connection; furthermore, the sender can make this determination for every ACK packet it receives. This is impossible to accomplish using RTT measurements, which do not allow a sender to differentiate between delay variations due to increases or decreases in the forward or reverse paths of a connection. As a result, Santa Cruz eliminates the need for Karn's algorithm [49] and does not require any RTO timer exponential back-off mechanisms that cause long idle periods over lossy links.

The congestion control algorithm introduced in Santa Cruz therefore allows the detection of the incipient stages of congestion, allowing a sender to increase/decrease its *cwnd* size in response to early warning signs. Given that the congestion control algorithm makes adjustments to the *cwnd* size based upon delays in the network and not on the arrival of ACKs in general, the authors suggest that the algorithm is robust to ACK losses.

The new error recovery mechanisms in Santa Cruz perform timely and efficient retransmissions of lost segments, avoiding unnecessary retransmissions for correctly received segments when multiple losses occur from a *cwnd* of data, and provide RTT estimates during periods of congestion and retransmissions. In addition, when multiple segments are lost per transmitted *cwnd*, it provides a mechanism to perform retransmissions without waiting for a RTO timer to expire.

The authors' simulation results show that Santa Cruz provides throughput improvements over TCP Reno and *TCP Vegas* [193] in two major areas, congestion control and error recovery. However, the bottleneck link in their simulation topology is not a true representation of a wireless link, where losses occur both in bursts and randomly, and where delays are highly variable on both the forward and reverse paths, especially when contention-based wireless networks are used.

The performance of Santa Cruz was later assessed for usage over wired-to-wireless scenarios by the authors of [194], who were also the first to implement the Santa Cruz code into the TCP stack of the Linux operating system. Some fundamental flaws were discovered with Santa Cruz in their experiments. Its estimation accuracy of network conditions was shown to be poor in the presence of multiple TCP connections along

the same path, and when there were additional delays over the wireless path due to local retransmissions taking place, thereby jeopardising any performance gains from its novel delay-based mechanisms. In addition, Santa Cruz was shown to be poor at discriminating between congestive losses and random losses, causing its performance to fall-back to the equivalent performance of TCP Reno under certain conditions. The authors then propose an enhanced alternative algorithm to Santa Cruz, which at best only solves the problems associated with multiple TCP connections in coexistence.

TCP Westwood+

Westwood+ [195] is a sender-side modification to TCP Reno's congestion control algorithms in an attempt to provide enhanced performance over heterogeneous paths. A Westwood+ sender uses an end-to-end *bandwidth estimator* (BWE) to alter the size of its *cwnd* and the *ssthresh* value, specifically after a segment loss event. Using a *low-pass filtering* mechanism, it monitors the rate of returning ACKs from the receiver. The measurement is used to adaptively decrease the *cwnd* size and *ssthresh* values after a congestion episode. In this way, Westwood+ substitutes the classic *multiplicative decrease* paradigm with an *adaptive decrease* paradigm. The idea behind this is to continuously be aware of the end-to-end bandwidth along a connection, so that at the point of a segment loss occurring Westwood+ sender is able to adaptively govern its *cwnd* size to better reflect end-to-end network conditions. Its authors state specifically that TCP Westwood+ can increase throughput over wireless networks.

A BWE sample calculation is performed for every RTT in Westwood+ by counting and filtering the flow of incoming ACKs. In summary then, Westwood+ leaves unchanged the probing phase of the legacy TCP but it substitutes the multiplicative decrease phase with an adaptive decrease phase, which sets the *cwnd* by taking into account the BWE value. As a result of this it has the potential to work well over wireless networks because it will reduce its sending rate severely only when a significant number of segments are lost. A single loss event due to an erroneous radio link therefore does not cause much harm [3].

In [196] the authors study the impact of uniformly distributed random segment losses on the *goodput* performance of TCP Reno and Westwood+ using a lossy bottleneck

link. It is shown that Westwood+ improves data goodput with respect to Reno when losses are non-congestive, with its *cwnd* size on average being four times larger than Reno's throughout a connection. Similarly, in [196] it is shown that Westwood+ improves on data goodput performance over TCP NewReno over channels with bursty segment losses. However, all experiments have been limited to simulation studies only.

In [197], the behaviour of Westwood+ is studied, particularly its BWE mechanism over varying random loss rates. The authors discovered that as the loss rate increased in their simulated experiments, the BWE estimate became more oscillating, which had a direct oscillating impact on the sender's *cwnd* size. This was due to the local retransmissions taking place over the wireless path, which consumes some of the bandwidth perceived by a TCP sender.

In [198] TCP Westwood+ performance is evaluated in the presence of segment losses caused by random BERs over a wireless link. The authors highlight that Westwood+ is unable to discriminate the causes of losses in a path, and suggest that its bandwidth estimation technique therefore incorrectly counts those segments that become corrupted whilst in transit. To counter this they propose *EW-TCP* as a sender-side enhancement to Westwood+, which has been designed to more accurately measure end-to-end bandwidth by counting corrupted data and acknowledged data separately per RTT. The authors use simulations to compare the performance of TCP Reno, NewReno, Westwood+, and EW-TCP over uniformly distributed loss conditions. Their simulation results revealed that a Westwood+ sender suffers from many RTO events due to segment losses caused by the wireless link. Specifically, as the corruption rate exceeded 0.5%, their EW-TCP outperforms Westwood+ by at least 8%, and up to 78% in some cases. In fact, for loss rates between 0.5% and 7% over the wireless path, Westwood+ performed on equal terms with the older NewReno.

However, the authors' simulation of the wireless link does not make use of local error recovery mechanisms; neither do they assess the impacts of contentions due to multiple devices in the wireless network, which are all typical characteristics of last-hop 802.11 WLANs. Real-world studies would provide more accurate insights into the RTO issues with Westwood+, and whether or not EW-TCP is superior.

TCP Veno

TCP Veno [199] is a proactive sender-side TCP implementation aimed specifically at data transmissions over lossy paths. It is a direct modification to the legacy TCP congestion control algorithms, with the objective of trying to improve end-to-end performance over wired-to-wireless paths by heuristically discriminating between segment losses caused by congestion in the wired path and random losses due to the wireless link in the path. Its functionality is designed to keep a TCP sender stable in the presence of randomly distributed segment losses that are typical of wireless paths, yet deal with traditional wired losses in a similar manner to the legacy TCP.

The key feature of Veno is that it monitors the network path for levels of congestion, and then uses this information to determine the cause of loss. If a loss occurs whilst Veno is in the congestive state, it assumes that the loss is due to network congestion; otherwise it assumes that the loss is random. In each of these situations Veno adjusts the *cwnd* size differently (i.e. the multiplicative decrease phase), taking into account that random losses require a less aggressive reduction of the sender's *cwnd* in order to maintain a higher throughput over the wireless portion of the connection. Another feature of Veno is its modified linear portion of the additive increase algorithm of the legacy TCP, so that when Veno is in the congestive state the *cwnd* growth is less aggressive. This effectively allows any self-induced network congestion to be relieved without segments being dropped in the wired network, helping Veno connections to maintain a higher overall throughput by remaining in an optimal operating region for longer.

Veno is able to determine whether or not it is in the congestive state by using calculations adopted from TCP Vegas [193] to determine the value of N, the number of backlogged segments at a bottleneck queue somewhere along the connection path. The value of N is continuously updated during a live connection, and is derived from: $N = (ExpectedRate - ActualRate) \times RTT_{min}$, where RTT_{min} is the minimum of all the measured RTT samples. Hence, if $N < \beta$ when a segment loss occurs then quite simply Veno assumes that the loss is more likely to be random (most likely occurring in the wireless link), than due to congestion. In [199] a value of 3 is suggested for β.

In essence, a Veno sender monitors the network and uses that information to decide whether losses are likely to be caused by congestion or by the random BERs of a wireless link.

With regards to its performance, the authors of [200] and [201] revealed from their experiments of running TCP Veno over WLANs that a performance improvement of up to 30% can be achieved under high wireless transmission errors.

In [199] a potential drawback of Veno is raised; its authors have not addressed the impacts of burst transmission errors that are typical of wireless channels. In a nutshell, it implies that Veno does not have inherent mechanisms to deal with the problem of multiple packet losses, but relies on previous techniques such as the SACK option. It is also shown in [199] that for random losses, a Veno sender often misinterprets them for congestive losses thereby reducing its *cwnd* drastically. In such situations the performance of Veno drops back to that of TCP Reno. The authors claim that it occurs due to false alarms when N is trapped in a congestive state, a possible weakness of the algorithm.

In [202], another improvement that could be made to TCP Veno has been suggested; the authors carried out experiments to investigate how competent Veno is at classifying the actual cause of segment losses. Their findings reveal that a better model can be applied to Veno in an attempt to minimise its misclassification percentage of up to 34.5%. The authors of [203] and [204] also conclude with similar problems being identified when there is heavy cross-traffic in coexistence with Veno connections across the Internet. In [204], the use of Veno with different TCP error recovery techniques is recommended to counter such problems. In [205] *TCP Veno+* has been proposed as an alternative algorithm in an attempt to improve on the accuracy of Veno's segment loss identification in light of heavy network congestion.

In [206], TCP Veno's loss distinguishing accuracy is evaluated extensively under different network parameters, and it is revealed that congestion loss accuracy and random loss accuracy are indirectly related to each other. The authors propose *TCP NewVeno* as a solution to these inefficiencies of Veno, and show via simulations that it can provide significantly enhanced performance. However, there were no studies

comparing the performance of Veno and NewVeno over a wireless path with random losses. Further to this, experiments using a real-world last-hop WLAN would provide more useful insights into both algorithms.

In [206] it is also observed that Veno still misses much of the available bandwidth of a last-hop wireless network when the network load is light and random losses are pervasive. This is due to Veno's indiscriminative reduction of the *cwnd* when random loss occurs. The authors suggest the implementation of a new variable called *congestion loss rate*, which can assist Veno in reacting more appropriately in response to random losses and consequently take advantage of more available bandwidth of a wireless link, without sacrificing TCP *fairness* or *friendliness* issues. However, the performance improvements are only verified via simulation studies using predefined random loss rates, which somewhat limits the amount of reliance that can be placed on the newly introduced variable.

In [207] it is shown that TCP Veno did not perform well in the presence of long disconnections of wireless devices from the AP in a WLAN. They also showed that TCP NewReno performed better than Veno for very high loss rates over the WLAN. However, all results are based on simulation studies only, and provide little insight into real-world behaviours.

Finally, based on the original evaluation of TCP Veno in [199], the suggestion of 3 as the optimal value for B could be an area of further investigation. It is also debatable whether or not there is an optimal value for the reduction factor in the size of the *cwnd* upon the detection of random losses, for which the authors recommend a factor of 4/5. In summary, there may be optimal values for differing wireless channel conditions and network types.

TCP Hybla

TCP Hybla [208] is a sender-side TCP proposal with the primary objective of providing better performance in end-to-end paths that possess larger segment RTTs due to wireless paths, for example a long-delay satellite radio link. Briefly, its core enhancements include a modification of the legacy TCP congestion control algorithms for higher latency paths, enforcing the use of the SACK option policy (to counter

multiple segment losses arising from larger *cwnd* sizes) and segment timestamps (to counter RTO timer exponential back-off issues due to higher RTTs), and the use of a *segment pacing* technique.

Hybla's modifications to the sender-side *cwnd* evolution is a direct result of analytical studies on its behaviour in such conditions, leading to the conclusion that altering the *cwnd*'s sole dependence on the RTT for growth is a viable solution. In essence, the higher the latency of a segment's journey, the larger the *cwnd* size needs to be in order to achieve a given throughput, generally calculated by *cwnd*/RTT. This leads to the rationale that for higher latency TCP connections the *cwnd* growth should be more accelerated to take advantage of end-to-end system bandwidth more effectively. In more detail, Hybla uses the notion of a parameter ρ as an equalisation term for the RTT_0 of a reference wired TCP connection and the actual RTT of a high latency connection. It then equalises the rate of injection of new data (i.e. how the *cwnd* grows) by using ρ to govern the *cwnd* evolution upon the arrival of ACKs.

In [209] and [210] it is shown via simulations and live testbed experiments that Hybla provides performance improvements over TCP Reno with SACK and TCP NewReno over a GEO-based satellite wired-to-wireless topology. However, although Hybla is shown to be superior over radio links with long delays and high loss rates, it would be insightful to determine its performance in a more generic heterogeneous environment. There appears to be a lack of literature assessing Hybla's extended usage over last-hop WLAN scenarios.

JTCP

JTCP [211] is a fairly recent *jitter-based* end-to-end TCP sender enhancement designed to be robust over heterogeneous/wired-to-wireless paths with higher random loss rates. The key idea in JTCP is to apply a *jitter ratio* (Jr) to the conventional TCP congestion control algorithms. The Jr is calculated according to the *interarrival jitter* per segment, which is defined according to the *real-time protocol* (RTP) [212]. The interarrival jitter is the time difference between two segments transmitted by a sender, and the difference between the same two segments arriving at the receiver. If the interarrival jitter is greater than zero then it implies that the second segment took longer to travel through the network than the first segment, which in turn implies that

some time was lost through queuing delays along the path. The Jr is therefore an important guide to determine whether TCP segments are being queued or not.

In JTCP a congestion event is defined as an event when triple DUPACKs are received by the sender and where the Jr is significant, which is calculated once per RTT. As a result, JTCP is able to distinguish the congestion-related losses and random losses.

The performance improvements offered by JTCP over other sender-side TCP variants has been shown via simulated experiments in [211] and [213]; JTCP offers considerable throughput gains over other versions of TCP when there are random losses along the path, and with varying delays over the lossy link. The limitations of the authors' simulations arise from the fact that they use a uniformly distributed error model for their wireless link, which is not a true representation of real-world conditions. The authors also provide little insights into the wireless protocol being used, so it is difficult to validate how JTCP would perform over a WLAN that has its own error recovery mechanisms.

In [213] a potential issue with JTCP was discovered over wireless paths with significantly high error rates. The authors suggest that in such conditions JTCP may not be able to distinguish wireless losses from congestive-losses via the Jr value. Hence further efforts have been suggested to investigate the validity of this claim.

In [207] a quantitative evaluation of JTCP using simulations is carried out alongside other sender-side wireless TCP proposals. The authors use both a *wide area network* (WAN) and a LAN topology, where the end-to-end path in each configuration contains two wireless links as opposed to just a single WLAN in the path. Again, the authors do not specify the wireless protocol being used, and do not consider local error recovery over the wireless link. The impacts on JTCP of random segment losses and device disconnections in the wireless links are studied, as well as the impacts of varying propagation delays over the wireless links. The simulation results revealed that JTCP was one of the top performers under varying random loss rates and disconnections over both topologies, highlighting its superiority in being able to discriminate between random and congestive losses. However, JTCP fell short by showing a significant drop

in its average throughput performance when competing for bandwidth whilst running alongside other TCP connections.

TCP-DCR

TCP-DCR (Delayed Congestion Response) [147] is a recent sender-side TCP modification for tolerating segment losses that occur over a last-hop wireless network that also incorporates its own link-layer local error recovery mechanism. The simple idea of TCP-DCR is to allow the wireless network to recover lost segments by itself, thereby limiting a sender's response to mostly congestive losses. This is achieved by waiting a certain delay period, τ, upon the reception of the first DUPACK before the TCP fast retransmit and fast recovery algorithms are invoked. A sender assumes that after waiting a period of τ for the confirmation of a successful transmission, the loss must be due to congestion.

Whilst waiting the τ period for the arrival of a new ACK, a TCP-DCR sender does not respond to incoming DUPACKs. In other words, the wireless protocol (i.e. the AP) has a τ time period to locally recover the lost segment that is being requested by the wireless TCP receiver, after which all losses are assumed to be occurring in the wired domain. If within the bounded τ time period the sender receives a new ACK, it proceeds with the next set of transmissions as though the loss never occurred. The delay in responding to actual congestion losses determines the overall performance of TCP-DCR, and the choice of τ is therefore a critical aspect in TCP-DCR implementations. There is a trade-off between unnecessarily inferring congestion and unnecessarily waiting a long time before retransmitting a lost segment. The authors suggest that the value of τ is set to the current RTT value.

The TCP-DCR delayed response mechanism only operates whilst in the congestion avoidance phase. The slow start phase is unmodified and operates as usual. Therefore, for short TCP flows where the majority of data sending is carried out during the slow start phase, such as with short-lived web traffic [44] [158], the TCP-DCR algorithm would not be active and provide any benefits.

The TCP-DCR implementation is sensitive to coarse timer granularities that exist within different operating systems, which can affect the accuracy of τ. If τ is not set

accurately enough, it can lead to RTO timer expirations before TCP-DCR gets the chance to take action.

One of the key assumptions made in the design and successful operation of TCP-DCR is that the wireless protocol uses a simple link-layer ARQ mechanism at the AP, which does not attempt in-order deliveries of segments. Unfortunately, some of the most popular last-hop wireless networks are based on the IEEE 802.11 MAC standard, which uses a positively acknowledging ARQ mechanism to deliver frames in-order. Of course this stop-and-wait approach by the 802.11 MAC would increase the RTT experienced by a TCP-DCR sender, causing the τ timer to expire. Since τ is less than the RTO timer value, the sender will reduce its *cwnd* size earlier than necessary.

In [147] and [214] the authors use simulation to study the performance improvements of TCP-DCR, and it is shown that it does indeed provide enhanced performance over a variety of error rates and delays over the wireless path. However, it would be more useful to see how TCP-DCR functions over a last-hop 802.11 WLAN with multiple devices contending for channel access, which is more representative of the real-world today. Additional delays caused by channel access contentions may also have an effect on the operation of TCP-DCR's delayed response function.

3.4.2.3 RTO Approaches

Sometimes, wireless network effects such as device disconnections from the AP or link-layer error recovery mechanisms can cause sudden delay spikes. Such an unexpected and sudden increase in delay affects the measured RTT of a TCP sender in the wired domain. A fixed TCP sender is not prepared for such an event, and is unable to adapt quickly enough to changing conditions along the connection [107]. It is usually the case that in such a scenario a TCP sender will be forced into a RTO timer expiry event, causing it to retransmit the delayed segments and drastically reducing its *cwnd* size to reinitiate the slow start algorithm. Since a RTO is reserved for cases of serious network congestion, more often than not, the timeout is spurious and unnecessary. Therefore, avoiding (though detection) or recovering from spurious RTO events quickly remains an important challenge for sender-side TCP implementations over wired-to-wireless paths [106] [148] [215].

Upon a spurious timeout a legacy TCP sender assumes that all outstanding segments are lost and retransmits them unnecessarily. It has been shown in [216] that the *go-back-N* retransmission behaviour of TCP triggered by spurious timeouts is due to the *retransmission ambiguity* problem, i.e. a TCP sender's inability to distinguish an ACK for the original transmission of a segment from the ACK for the retransmitted segment.

Shortly after a spurious timeout at the sender, ACKs for the original transmissions will arrive. On reception of the first ACK after the timeout, a sender must interpret this ACK as acknowledging the retransmitted segment, and must assume that all other outstanding segments have also been lost. Thus, the sender enters the slow start phase, and retransmits all outstanding segments. The go-back-N approach triggers the next problem, where the receiver will generate a DUPACK for every segment received more than once. It has to do this because it must assume that its original ACKs may have been lost. The reception of DUPACKs at the sender may trigger a *spurious fast retransmit*, another widely known problem [148].

Eifel Detection Algorithm

The *Eifel Detection Algorithm* (EDA) suggested in [217] and recommended by [218] is a simple yet effective idea that allows a TCP sender to detect whether or not a RTO event is necessary. EDA works in conjunction with the timestamps option [109] to reliably detect spurious timeouts, with the aim of preventing unnecessary retransmissions and activation of congestion control procedures. Timestamps allow a TCP sender to determine which segment is being acknowledged by an incoming ACK, the original transmission or the retransmission. EDA therefore solves the issues related to the aforementioned retransmission ambiguity problem at the sender.

Including the TCP timestamp option field in every segment and ACK is not without cost; the 12 bytes overhead per field can be seen as quite heavy. The advantage of using the timestamp option is that it is already widely deployed in the Internet [44].

In [217] it is shown that in a lossless path, EDA improved the performance for all TCP variants under a variety of delay spikes. However, wired-to-wireless paths are certainly not lossless. In reality delay spikes over wireless paths are often accompanied with lost data segments and ACKs, which can further exacerbate the situation at the sender. In

the extreme case, all but the oldest outstanding data segments can be lost. The EDA proposal simply specifies that the transmission after detecting a spurious timeout always resumes with the next unsent segment. This works fine when none of the delayed segments are lost, but over wireless paths consecutive segments can be lost. Simply transmitting new data in this case leads to a second genuine RTO event, as discovered in [148]. A further limitation of EDA is that it is unable to deal with spurious timeouts that occur during a fast retransmit and fast recovery procedure [105].

F-RTO

The *forward RTO* (F-RTO) [150] is a sender-side TCP modification for recovering from spurious timeouts, without requiring the use of any TCP options to operate. F-RTO uses a set of simple rules to avoid unnecessary retransmissions in the event of spurious timeouts at the sender.

It works as follows: after retransmitting the first unacknowledged segment triggered by a timeout, the F-RTO algorithm at a TCP sender monitors incoming ACKs to determine whether the timeout was spurious and to decide whether to send new segments or retransmit unacknowledged segments. The algorithm starts by transmitting new segments after a timeout and reverts to standard go-back-N behaviour only if a DUPACK is received. Otherwise, the timeout is considered spurious and the sender continues transmitting new data.

In [150] F-RTO has been shown to improve TCP performance in comparison to other TCP variants over a variety of delay and loss conditions. However, F-RTO is unable to accurately classify timeouts caused by severe segment reordering that can occur over wireless networks. In such cases it reverts to the behaviour of a regular TCP implementation. F-RTO is also unable to revert to a previous congestion control state if the RTO was indeed spurious [105]. In [105], [219], and [220] it is argued that the nature of the spurious timeout problem calls for dynamic modifications to the RTO timer itself during a connection, as opposed to simply responding after the RTO timer has expired, and respectively *PH-RTO, CA-RTO*, and *WB-RTO* are proposed as proactive alternatives to the more reactive F-RTO.

Finally, there appears to be a lack of literature relating to the experimental evaluation and true effectiveness of F-RTO in realistic conditions, such as using a real-world last-hop 802.11 WLAN over a TCP connection.

3.4.2.4 Loss Differentiation Algorithms

For many years now researchers in the field have attempted to design a TCP that is able to determine the cause of errors over heterogeneous paths by using heuristics within TCP based on its perception of network conditions [221] [222] [223]. More recently, the field has been introduced to the concept of *loss differentiation algorithms* (LDA) that can be used by transport protocols such as TCP for improving end-to-end performance in heterogeneous environments where segment losses also occur randomly, as well as in traditional congestion-related ways [224] [225].

In a nutshell, a LDA implementation can assist a TCP sender with the *Boolean* classification of the cause of problems in the network, which is based on a function of the state of TCP variables and sender-side measurements of the network. LDAs usually decide whether the network path is congested, or not congested, which can be used to deduce the cause of a lost segment and/or unexpected delays. Hence, LDAs can assist TCP in deciding which direction to move its congestion control mechanisms after each transmission [226].

Non-Congestion Packet Loss Detection

One of the earliest LDA proposals to be used in conjunction with a sender-side TCP to differentiate between the causes of segment loss in a wired-to-wireless environment is the *non-congestion packet loss detection* (NCPLD) scheme [227]. NCPLD is an end-to-end scheme designed for TCP connections that have a last-hop wireless link. The simple idea in NCPLD is that it maintains the minimum RTT throughout a connection, so that when a segment loss occurs, if the current RTT is close to the minimum RTT then the loss is assumed to be due to the wireless link and appropriate error recovery mechanisms are invoked whilst maintaining the *cwnd* size. However, NCPLD was only ever evaluated in a simulation environment, so its applicability in the wider Internet remains inconclusive. Recent simulation studies in [226] have revealed that the original NCPLD scheme does not exhibit high accuracy either, mainly due to the

coarseness with which the TCP sender measures the RTT, and an *enhanced* NCPLD algorithm is proposed. The original NCPLD causes a TCP sender to correctly detect congestion losses, but tends to regard most of the segment losses to be congestion-based losses [225].

Another LDA scheme that utilises the same idea as NCPLD is *TCP-Probing* [228], where the detection of segment losses triggers the sending of probes into the network to measure the current RTT. The TCP-Probing implementation however has high processing overheads which can prove costly at high loss rates where segment losses occur frequently.

Packet Loss Pairs and Hidden Markov Models

In [229] the authors combine two techniques, *packet loss pairs* (PLP) and *hidden markov modelling* (HMM), to develop a LDA that can assist TCP senders in hybrid wired-to-wireless environments. The LDA proposed is able to perform end-to-end differentiation between wireless and congestion losses. A PLP is a pair of segments sent back-to-back by the sender such that exactly one of them is lost along the journey. Since the two segments travel closely enough together in the network up to the point where one of them is lost, the segment that is not lost is then able to convey the RTT observed at the point of segment loss back to the sender. HMMs are a powerful modelling tool for applying a mathematical framework to a wide range of applications in order to gain theoretical understandings of outcomes and behaviour. In this context, HMMs are typically used to model network conditions accurately, being able to encapsulate the effects of lossy and non-lossy links using probabilities associated with being in each state. A TCP sender is then able to infer the type of segment loss by referring to the HMM state that best fits the RTT observed at the point of loss.

Unfortunately, the concept of PLPs was originally intended for wired paths only [230], and their usage of wireless networks may render them ineffective. To illustrate, in IEEE 802.11 WLANs, the AP would not be able to send two segments back-to-back and guarantee that they both will take the same journey over the radio link and both experience the same delays. With the CSMA/CA mechanism at the MAC, there could be a significant gap between the sending of each segment due to a host of reasons. In addition, there can exist two bottleneck points in a wired-to-wireless path; one

occurring in the wired network at intermediate routers, and one occurring at the gateway to the wireless network (i.e. at the AP). However, the effectiveness of PLPs relies on there being only one most congested point in the path, where losses and delays are most significant compared to other parts of the path [230]. Finally, using HMMs to model the entire distribution of RTTs over a wired-to-wireless path in the real-world is a challenging and complex task. When using 802.11 WLANs for example at the last-hop, many factors can determine the transmission delays experienced by TCP segments traversing the wireless hop. The number of devices in the WLAN can increase contention delays; the stop-and-wait ARQ mechanism can increase instantaneous delays; and buffering of segments inside the AP can increase queuing delays. Each of these can occur individually or concurrently, thereby complicating the deterministic nature of using HMMs to model RTT outcomes. In [230] the authors only use simulations to validate their hybrid LDA, without using a wireless link in their topology. They only simulate the effects of a wireless link to test their algorithm.

ZBS Hybrid Scheme

In [224] the authors evaluate two previous LDA proposals, the *biaz scheme* [231] and the *spike scheme* [232], and then go on to propose their own LDA, the *zigzag scheme*, after discovering weaknesses in them both when used with TCP over wireless paths. After evaluating their own zigzag scheme, they go one step further and develop a hybrid LDA algorithm by the name of *ZBS*, which combines the strengths all three schemes into one. Unfortunately, the ZBS algorithm was only ever tested using simulation studies; and the authors state that they assumed an error-free reverse channel for their wireless link, i.e. TCP ACKs are not lost, which is not a true reflection of data transmissions over radio links. Although the ZBS algorithm may be a strong contender in the LDA arena, its needs to be evaluated in a more realistic manner using a real-world network testbed with an operational last-hop wireless network. Finally, the computational complexity of the ZBS algorithm in differentiating the cause of losses is also questionable in high delay/loss environments, as it involves more addition and multiplication operations per segment arrival/loss than all the other schemes individually. Again, in the real-world this would need to be evaluated by measuring the responsiveness of a TCP sender using the ZBS algorithm. It has been reported in [177] that ZBS still suffers from loss discrimination problems due to the inherent weaknesses of each of the three LDAs that it consists of.

TCP Westwood with Bulk Repeat

A fairly recent attempt at combining existing LDA mechanisms with a popular TCP sender-side implementation has been proposed in [233]. Here the authors take the existing TCP Westwood [234] variant and combine it with a *bulk repeat* algorithm that works hand-in-hand with the spike [232] and *rate gap threshold* (RGT) [227] LDAs to tackle the problems associated with the original TCP Westwood in heavy loss wireless environments. They name their proposal *TCPW BR*.

The BR algorithm requires three modifications to a sender-side TCP Westwood implementation, although the authors do suggest that the enhancements could be applied to any TCP variant in theory. BR includes the *bulk retransmission* mechanism, a *fixed retransmission timeout* value, and an *intelligent window adjustment* system.

Bulk retransmission is used to retransmit all outstanding unacknowledged segments from a *cwnd* whilst in the fast retransmit phase. This is done when an ACK arrives that only partially acknowledges new segments from the *cwnd*, implying that more than one segment may have been lost from the *cwnd*. To avoid recovering individual segments once per RTT, a bulk retransmission efficiently recovers all segments in a single RTT. Obviously the number of bulk retransmissions to perform is dependent on whether the losses are congestion related or wireless related, as the former situation would need a less aggressive approach in order to relieve the congestion.

The fixed retransmission timeout strategy used in BR on the notion that in high loss wireless environments, the same segment may be retransmitted several times due to constant RTO events before success. According to Karn's exponential back-off algorithm [216], it would severely stall a TCP sender for long periods between retransmissions. Hence the authors propose a 'freezing' of the RTO timer for wireless losses only, which prevents the timer from being doubled each time a segment retransmission takes place due to non-arrival of an ACK.

The intelligent window adjustment mechanism of BR is a simple idea that basically leaves the size of the *cwnd* unchanged when segment losses occur over the wireless path of a connection, as opposed to reducing it and dropping back the sending rate.

This allows more segments to be transmitted per RTT by the sending TCP for wireless losses.

The three concepts described in BR rely on the accurate discrimination of wireless losses from congestion losses by the TCP sender. The authors achieve this by combining the spike and RGT LDAs to form a hybrid that works well at both low and high error rates. Spike keeps track of the minimum and maximum RTT of a connection, and feeds the values into RGT which calculates the gap between the expected rate and the actual rate. If the gap is larger than a predefined threshold then all losses are assumed to be congestion related, otherwise all losses are assumed to be occurring over the wireless path.

In [233] the authors use simulations to evaluate the throughput performance of TCPW BR in comparison to TCP Westwood and TCP NewReno. They use a last-hop wireless link in a wired-to-wireless topology, and experiment with varying uniform error rates and bursty error conditions over the wireless link. Their results reveal that TCPW BR performs significantly better than NewReno under all uniform loss conditions, and improves on Westwood's performance when error rates exceed 5% over the wireless link. Under bursty loss conditions, TCPW BR significantly outperforms NewReno and Westwood for all error rates up to 50%. However, the authors make one critical assumption in the simulation of the wireless link: they do not use local link-layer retransmissions over the wireless channel. In an 802.11 WLAN, this would be the equivalent of ignoring the complex interplay between MAC layer retransmissions and the error recovery mechanisms of the TCP sender. This is not a realistic method in which to test a TCP proposal for usage in today's heterogeneous climate, where 802.11 WLANs are very popular at the last-hop path of TCP connections across the Internet. The authors also do not comment on whether or not losses occur in both directions over their simulated wireless link.

Recently, it has been shown [225] via simulation studies that the parameters a=0.4 and b=0.05 used for the spike LDA in TCPW-BR yield a poor classification of wireless losses in comparison to congestion losses. The authors of this work therefore recommend using a=1/2 and b=1/3 as more suitable values.

TCP-RoS

A recent attempt at combining a LDA with an existing TCP variant is proposed in [177], in which the authors extend TCP-Casablanca [6] so that it is able to differentiate between congestion losses and wireless losses. The authors name their extended version as *TCP-RoS*, and show that it improves performance over heterogeneous paths in comparison to the original. Although the TCP-RoS scheme isn't strictly a sender-side TCP modification (it requires modifications at intermediate routers and to the TCP implementation at the receiver), it has been mentioned here because it is one of the most recent attempts at using an LDA in conjunction with TCP for improving wired-to-wireless performance.

LD-LogWestwood+ TCP

Another recent attempt at combining a LDA with an existing TCP sender-side variant is proposed in [235], to which the authors give the name *LD-LogWestwood+ TCP*. It results from combining the spike LDA scheme [232] with the recently proposed *TCP LogWestwood+* [236], which itself is a direct modification to the original TCP Westwood+ [195] that implements a logarithmic evolution of the *cwnd* size in the congestion avoidance phase to replace the linear function. The authors discovered through simulation studies [237] that LogWestwood+ was not aggressive enough for higher loss rates over the wireless path, and felt that it needed a way of discriminating between congestion losses and wireless losses in order to alter the *cwnd* size appropriately and achieve peak performance over wired-to-wireless paths. The spike LDA is implemented and used to inform the sender of the most likely cause of segment losses. Typical settings are used for the spike LDA in LD-LogWestwood+ (a = 0.4 and b = 0.05).

Segment losses in LD-LogWestwood+ are detected without any assistance from the spike LDA. Upon detecting a congestion-related loss, LD-LogWestwood+ performs the congestion avoidance phase in a similar fashion to LogWestwood+, albeit with a slightly less aggressive *cwnd* evolution. Upon detecting a wireless loss, LD-LogWestwood+ continues to increase the size of the *cwnd* in order to take advantage of unused wireless bandwidth, assuming that the wireless path has its own error recovery mechanism to recover the lost segments, although the authors do not comment on this aspect.

Through simulation studies, the authors evaluate LD-LogWestwood+ alongside previous wired-to-wireless TCP proposals and show that they perform very well over a range wired-to-wireless scenarios and varying loss rates. However, there is a lack of details as to whether the loss model for the wireless link affects reverse channel traffic, or only traffic on the forward channel.

The authors also use a fixed delay for the wireless hop in their simulations, which is somewhat misrepresentative of real-world wireless networks, i.e. 802.11 WLANs possess highly variable delays. The authors use 16 TCP receivers in the wireless last-hop, and maintain a static delay. In the real-world, a large number of wireless devices in an 802.11 WLAN would be contending for medium access, which would increase overall delays dynamically. Further, forward and reverse channel TCP traffic can suffer from problems associated with 802.11 WLANs (as mentioned in Chapter 2), such as unfairness, self-collisions, and the capture-effect, which would all contribute to higher segment loss rates and further impact on delays for a TCP sender in the wired domain.

The key assumption in LD-LogWestwood+ is that there is no congestion in the network upon a new TCP connection being initiated, so that the spike LDA's initial minimum RTT measurement isn't distorted to an unnecessarily high value. Since spike uses the minimum RTT to set the *stop* and *start* RTT thresholds for indicating congestion in the network, initial high values of the RTT can distort the threshold calculations in the short term. This could be significant for short TCP flows that transfer only small amounts of web traffic; it means that a LD-LogWestwood+ sender would see all segment losses as wireless losses if the RTT were to suddenly return to a slightly lower value (with congestion still imminent) before the short TCP flows were completed. For TCP segment losses over the wireless path, LD-LogWestwood+ does not decrease the size of the *cwnd*, but rather keeps increasing it in size. Such aggressiveness in the presence of congestion would further exacerbate the network congestion, leading to further segment losses. Hence, the stability of the LD-LogWestwood+ proposal relies on this potentially significant assumption.

3.5 Chapter Conclusions

This chapter has provided an up-to-date review of the knowledge and current state of the art in the field of TCP performance issues and enhancements over wired-to-wireless paths, including a review of some the most promising approaches and proposals to date. There has been a particular focus on i) those TCP enhancements that can be made easily at the sender-side on servers across the Internet, and ii) the issues and impacts on TCP senders of using the IEEE 802.11 WLAN standard as the wireless path for TCP connections, particularly over the last-hop portion of the journey.

The significance of these in combination will be assessed in the chapters to follow in the form of comprehensive experimental work in an attempt to advance this vast knowledgebase above and beyond what is already known. This thesis will attempt to extend the current state-of-the-art by bringing together the two distinct protocols, TCP and IEEE 802.11, and exploring any relationships between them in a wired-to-wireless configuration with the hope of generating new findings and making performance improvements.

Chapter 4

TCP Sender Resilience to BERs over 802.11 Channels

4.1 Introduction

One of the key characteristics of radio channels is that they possess significantly higher bit error rates (BERs) than traditional copper transmission media; 802.11 WLAN channels are no exception here. The fundamental problem for TCP connections that traverse such links is that a TCP sender in the wired domain is unable to classify the cause of segment losses. It cannot discriminate between losses occurring in the WLAN or losses in the fixed network, and therefore regards them both as network congestion by activating anti-congestion procedures.

Fortunately, the 802.11 WLAN MAC protocol implements its own error recovery mechanism which can deal with the higher BERs locally by transparently retransmitting erroneous TCP segments encapsulated by 802.11 frames. However, the MAC cannot be too persistent with its stop-and-wait retransmission mechanism to avoid unnecessary delays, so typically all 802.11 implementations will have a *retry limit* on the maximum number of attempts that can be made.

In situations where channel conditions are very poor due to significantly higher BERs, it is likely that all retransmission attempts by the 802.11 MAC will be unsuccessful, and hence the consequences on a TCP connection are devastating. First of all, the delays caused by retransmitting up to the retry limit will cause long delays for TCP users, and secondly this will lead to either a RTO event at the sender, or it will lead to a fast retransmission. Both events are severely degrading to end-to-end performance, as a TCP sender will reduce its sending rate by cutting the size of its *cwnd* for each occurrence of such an event. Therefore, understanding how resilient a TCP sender is to BERs over 802.11 WLAN channels is useful information, as it can help researchers to assess whether or not TCP needs more advanced error recovery mechanisms, and in which conditions they should be introduced [31]. To clarify, the BER is calculated by

dividing the number of erroneous bits received by the total number of bits transmitted [119].

In this chapter a wired-to-wireless TCP simulation environment is presented using the increasingly popular OPNET Modeler™ simulation suite, which incorporates full implementations of TCP and the IEEE 802.11 WLAN protocol. A customised WLAN independent channel bit error model is implemented to replace the standard uniform model, which allows subjecting a TCP sender to varying BERs over the last-hop 802.11 WLAN to determine its resilience in wired-to-wireless path. A series of experimental results are then presented evaluating the performance of popular applications that run over TCP, such as *FTP* and *HTTP*.

4.2 Background Information

Since the BER is the most widely quoted and mentioned problem associated with wireless radio channels, it is of paramount importance that researchers have accurate knowledge of the BER values that are problematic for TCP over 802.11 WLANs specifically. Many studies have been undertaken in recent year to assess the transmission error characteristics of wireless channels [69] [73] [74] [75] [76] [130] [238] [239] [240]; however each of these studies focus entirely on the error rates at the wireless packet level, as opposed to at the bit level. Further, these studies do not focus on the impacts of wireless error rates on wired protocols such as TCP, which are the ones that suffer the most.

Operating at the bit level when studying wireless packet errors allows for greater precision when assessing the impacts (or sensitivity) of multi-path fading effects due to the radio medium on packets in flight. This includes capturing bit errors that are correlated, and those that occur in bursts i.e. a contiguous sequence of bits are corrupted within the packet. So, although packet level loss rates of radio channels can be insightful to researchers, particularly for experiments that focus specifically on higher-layer protocols, operating at the bit level can introduce greater realism into experiments because it is how packet errors occur in the real-world [25].

A typical BER value often quoted for wireless channels in general is in the region of 10^{-5} [58] [241] [242], with some reports quoting values as high as 10^{-4} [140] [243] and 10^{-3} [27]. A BER of 10^{-3} implies that 1 in every 1000 bits received is corrupted within a wireless packet. A TCP segment encapsulated by a wireless packet would have a size of at least 1500 bytes including all added headers (actual TCP data size is 1460 bytes), which equates to a minimum of 12000 bits per packet. Assuming a uniform distribution of bit errors, a BER of 10^{-3} would cause on average 12 bit corruptions per received TCP segment (note that this does not apply to correlated bit errors that can occur within a packet). Recall from Chapter 2 that the IEEE 802.11 MAC does not utilise any FEC; hence a single bit corruption inside a frame can cause the frame to be discarded completely by the receiver. Each discard will then need a local 802.11 retransmission up to the retry limit before the MAC can proceed with the next in-sequence frame.

Through a survey of literature in the area, work focussing on the correlation between the BER of 802.11 WLAN channels and its impacts on TCP performance could not be located. One of the key difficulties arises from not being able to experiment in an environment where the BER is controllable. This chapter therefore aims to introduce some base-line results to the area.

4.3 OPNET Modeler™ Network Simulator

In this section the OPNET Modeler™ v14.0 network simulator package is introduced, alongside descriptions of the TCP and IEEE 802.11 WLAN implementations that it uses.

4.3.1 OPNET Modeler™

The OPNET Modeler™ commercial discrete event simulator was chosen as the platform for all simulation in this chapter because it offers excellent models and full support for the modelling and simulation of 802.11 WLANs, as well as for studying legacy protocols such as TCP [244]. Seen as a leading simulation package globally, the use of OPNET Modeler™ for studying TCP over wired-to-wireless paths has gained popularity by the research community in recent years [169] [245] [246] [247] [248].

Specifically, it has been shown in [249] that OPNET Modeler™ is more accurate at simulating TCP networks than other freely available simulators such as NS-2.

4.3.2 TCP Implementation in Modeler™

The implementation of TCP in OPNET Modeler™ is very detailed and has been based on all the latest publications and RFC documents relating to the requirements of a modern-day TCP [250]. Its comprehensive nature allows the selection of various 'flavours' of TCP at the sending side, i.e. at the server, with the option to select many extensions and options that are typical of TCP implementations in the real-world today.

4.3.3 IEEE 802.11 Implementation in Modeler™

The implementation of the IEEE 802.11 WLAN standard in OPNET Modeler™ is based on the original working specification published by the IEEE, offering full support for MAC and PHY specifications, for both 802.11b and 802.11g [251]. The OPNET Modeler™ WLAN suite uses the MAC DCF contention-based mechanism for infrastructure-mode WLANs, using exponential back-off procedures according to the CSMA/CA protocol, mimicking the real-world behaviour of the protocol as closely as possible.

4.3.3.1 WLAN Channel Model

The radio transmission channel used by 802.11 WLANs in OPNET Modeler™ is modelled by the *radio transceiver pipeline* (RTP), which is used to mimic wireless transmissions of frames. The RTP consists of fourteen individual stages (0 to 13) that feed into each other, where each frame transmission per device must traverse through the stages in sequence, as illustrated in Figure 4.1.

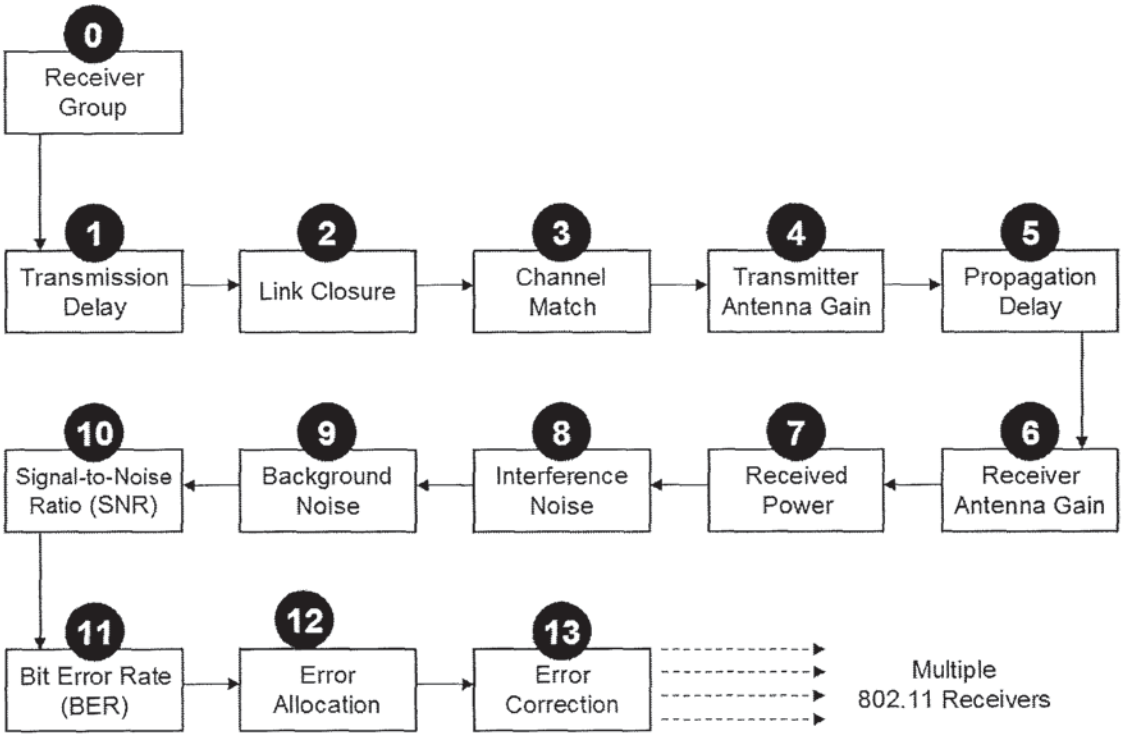


Figure 4.1: OPNET Modeler™ Radio Transceiver Pipeline (RTP) Stages

It can be seen from Figure 4.1 that each stage has a unique function that is associated with radio channel transmissions. For example, *stage 3* ensures that there is a channel match between the transmitter and the receiver, *stage 5* is used to compute the wireless propagation delay between the transmitter and the receiver, *stage 8* introduces the effects of interference noise to a frame transmission along the RTP, and so forth [251].

4.4 Simulation Environment

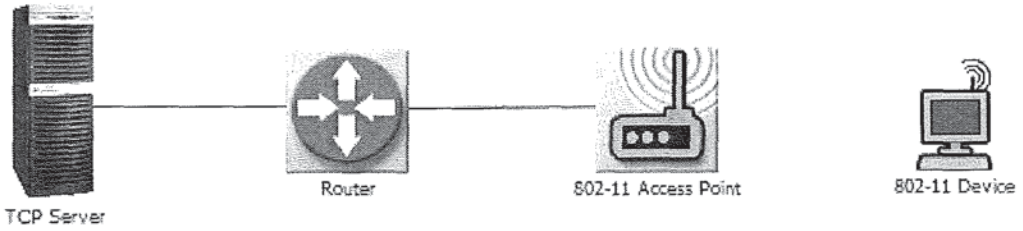


Figure 4.2: The wired-to-wireless topology used in OPNET Modeler™ v14.0

In this section the OPNET Modeler™ simulation environment and protocol settings that were used are clearly outlined.

4.4.1 Scenarios and Settings

The simulation scenario used was a typical wired-to-wireless topology, as shown in Figure 4.2, which consisted of a wired TCP server connected via an intermediate router to an 802.11 device in the WLAN via the AP. The server is assumed to be in the fixed Internet somewhere.

The TCP server was configured to support a host of applications, including both FTP and HTTP services. Since HTTP and FTP are both popular TCP applications used across the Internet by wireless end-users, this was an ideal way of evaluating TCP performance. The server was configured to use the popular Reno variant of TCP with a *ssthresh* value of 64 Kb. Figure 4.3 provides details of the TCP settings and options that were used, trying to be reflective of typical TCP Reno implementations in the Internet today.

Attribute	Value
Version/Flavor	Reno
Maximum Segment Size (bytes)	1460
Receive Buffer (bytes)	65535
Receive Buffer Adjustment	Windows Based
Receive Buffer Usage Threshold (of RCV BUFF)	0.0
Delayed ACK Mechanism	Segment/Clock Based
Maximum ACK Delay (sec)	0.200
Maximum ACK Segments	2
Slow-Start Initial Count (MSS)	2
Fast Retransmit	Enabled
Duplicate ACK Threshold	3
Fast Recovery	Reno
Window Scaling	Enabled
Selective ACK (SACK)	Enabled
ECN Capability	Disabled
Segment Send Threshold	MSS Boundary
Active Connection Threshold	Unlimited
Nagle Algorithm	Disabled
Karn's Algorithm	Enabled
Timestamp	Enabled
Initial Sequence Number	0
Retransmission Thresholds	Attempts Based
Initial RTO (sec)	3.0
Minimum RTO (sec)	1.0
Maximum RTO (sec)	64
RTT Gain	0.125
Deviation Gain	0.25
RTT Deviation Coefficient	4.0
Timer Granularity (sec)	0.5
Persistence Timeout (sec)	1.0

Figure 4.3: OPNET Modeler™ TCP settings used by the server

The TCP settings at the wireless device were set to those of the *MS Windows XP* stack, as offered by OPNET Modeler™. This is to reflect a typical end-user device running popular operating systems such as Windows XP. The TCP buffer sizes at both ends were set to their maximum possible values in order to allow the network to dictate TCP behaviour, as opposed to being limited by end-hosts.

The wireless path for TCP connections was configured as an 802.11b WLAN operating in infrastructure mode, offering a maximum data rate of 11 Mbps over the WLAN. The WLAN consisted of only a single 802.11 device situated a distance of 20 metres from the AP, as the aim of the work is to concentrate solely on channel error conditions as opposed to the effects on TCP of channel-access contention related issues. For the purposes of our experiments, the wireless device remained stationary at all times.

The 802.11 AP was configured to use a maximum size receive buffer of 128000 bytes to avoid situations of buffer overflows, which would distort the results. The frame retry limit on both the AP and the 802.11 device was set to the default value of 7, as specified by the 802.11 MAC standard. All wired connections were made using *Ethernet 100BaseT* links that were assumed to be error free. The one-way propagation delay between the wired server and the AP was set to 50 ms, and the intermediate router was configured to enforce a *drop-tail* queuing policy, with its buffer size set to the *bandwidth delay product* of the wired path.

HTTP Specification	Version 1.1
Web-page Interarrival Time (s)	uniform(15,60)
End-user Repeat Probability	Web Browsing Mode (Poisson)
Number of Web-pages per Request	constant(10)
Number of HTTP Objects per Web-page	Uniform(3,7)
Size of HTTP Objects (bytes)	uniform(500,1000)

Table 4.1: HTTP web-page request settings for 802.11 end-device

To simulate the effects of a wireless end-user accessing Internet services, the 802.11 device was configured to make HTTP web-page requests from the wired server, as well as perform FTP file download requests. The HTTP request settings were based on the results of comprehensive studies carried out in [252]. Table 4.1 provides details of

the application-specific settings that were used by the end-user making web-page requests via HTTP.

For the FTP file download requests from the TCP server, only single-flow connections were used between the sender and the receiver. The 802.11 end-user was configured to make fixed-size 15 Mb file downloads from the TCP server in the wired domain to simulate the effects of downloading a file from the Internet, and TCP segments having to traverse a wireless link in the last-hop portion of the connection.

4.4.2 Implementation of Custom BER Generator

The standard BER stage uses the SNR value computed in the preceding stage to perform a *modulation table* 'look-up' corresponding to the 802.11 PHY modulation scheme and current data transmission rate being used. Each SNR table look-up returns a BER value (set in the global variable *OPC_TDA_RA_BER*) that is then passed onto *stage 12*, which is responsible for applying the actual bit errors to frames using a random process.

The standard 802.11 WLAN channel model in OPNET Modeler™ does not allow users to inject custom bit errors over transmissions traversing the radio medium. After some thorough investigations, it was established that OPNET Modeler™ assumes an error free radio channel at all times, unless a wireless device moves out of range of the AP completely, which is 300 metres. Therefore, to capture the impacts on the TCP server of varying BERs over the WLAN a *custom BER generator* (CBG) was developed to replace the standard BER stage of the RTP, i.e. *stage 11*.

The custom CBG that was implemented overrides *stage 11* by manually setting the *OPC_TDA_RA_BER* variable before it is read by *stage 12*. This allows users to enter BER values in the range 0 to 1 via extended attributes of the *node model* of 802.11 devices in the user-space of the OPNET Modeler™ project editor. These attribute values are passed to the OPNET Modeler™ kernel space at runtime, and are used directly by *stage 12* to simulate the appropriate number of bit errors to be applied per frame. Hence, the CBG allows random bit errors to be artificially induced into each 802.11 frame that passes over the wireless channel.

4.5 Simulation Results and Discussions

This section presents the results of simulations that were performed with TCP Reno over a wired-to-wireless network. Each simulation per BER value was performed three times, with the average results presented here. The only variable parameter in all experiments was the BER value over the last-hop wireless path, where all bit errors were randomly induced. The BER values chosen for the experiments ranged from 10^{-8} up to 10^{-3}, representing good to bad channel conditions which are typical of wireless networks based on a review of literature, as discussed in a previous section. This range covers very good channel conditions to very poor conditions, and will give good insights into how TCP applications perform over the range.

4.5.1 TCP Server Congestion Window Behaviour

Initially the behaviour of the TCP server's *cwnd* is assessed, as it provides good insights into its sending performance and overall stability. To make a direct comparison of the *cwnd* evolution between a fully wired path and a wired-to-wireless path, some preliminary simulations were carried out. In the fully wired topology the 802.11 AP was replaced with a network hub, to which a TCP receiver was connected using Ethernet 100BaseT links. The wired TCP receiver had identical settings to the 802.11 device. For the wired-to-wireless path, the topology shown in Figure 4.2 was used, and a BER value of 10^{-8} was set as it represents good channel conditions over the WLAN. The TCP receivers in both scenarios used FTP to download an arbitrarily large file from the server, and simulations were run for 180 seconds each. The delay over the wireless path was the default delay as governed by the 802.11b RTP within the simulation environment, which was set to a data rate of 11 Mbps using the *additive white Gaussian noise* (AWGN) model setting for the radio channel. Figure 4.4 presents the results from OPNET Modeler™ that shows the evolution of the *cwnd* at the TCP server for each scenario, which reflects sending performance.

As can be seen from Figure 4.4, there is a noticeable difference in the evolution of the *cwnd* between the two types of paths for a TCP sender. The wired path *cwnd* advances more aggressively and by the end of the simulation has reached a size of 350 Kb. The *cwnd* in the wired-to-wireless path has a much less aggressive rate of growth, thereby

achieving a size of just 250 Kb by the end of the simulation, which is almost a 29% drop than in the equivalent wired scenario. Recall from a previous chapter that the size of the *cwnd* is directly proportional to the amount of data that can be sent per RTT, and therefore translates into a high sending rate.

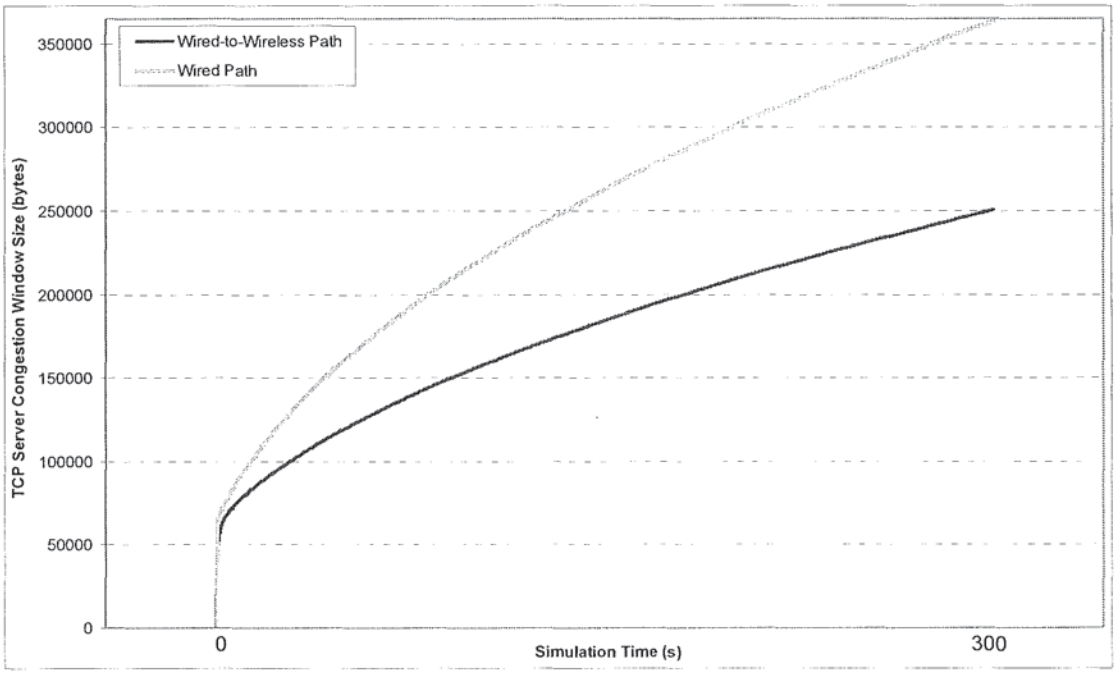


Figure 4.4: Comparison of *cwnd* evolutions for wired versus wired-to-wireless TCP paths

This simple simulation therefore highlights the effects of having a wireless path in the journey of TCP connections. The 802.11 WLAN will cause additional delays to TCP segments travelling over the radio channel. Further delays will be caused by local retransmissions taking place by the MAC. In summary then, even in good channel conditions, a last-hop 802.11 WLAN still cannot match the performance of a wired equivalent path. The difference between the two plots in Figure 4.4 is caused by the additional delays and retransmissions caused by the wireless path.

Moving onto the wired-to-wireless simulations, in Figure 4.5 the evolutions of the *cwnd* for BER values ranging from 10^{-8} to 10^{-5} are plotted for arbitrarily sized FTP file downloads (all plots are overlapping). As can be seen, all plots possess typical characteristics of a TCP sender, i.e. the initial slow start phase followed by the congestion avoidance phase, which takes over when the *cwnd* size exceeds 64 Kb. This plot indicates that the TCP sender in the wired domain is unaffected by wireless

channel conditions over the WLAN for BERs as high as 10^{-5} inclusive. This can be explained by the fact that the 802.11 MAC is able to perform its local error recovery of erroneous TCP segments within an adequate time interval, shielding such conditions from the sender. The sender is therefore able to perform the slow start and congestion avoidance phases without interruptions being caused by losses or RTO events.

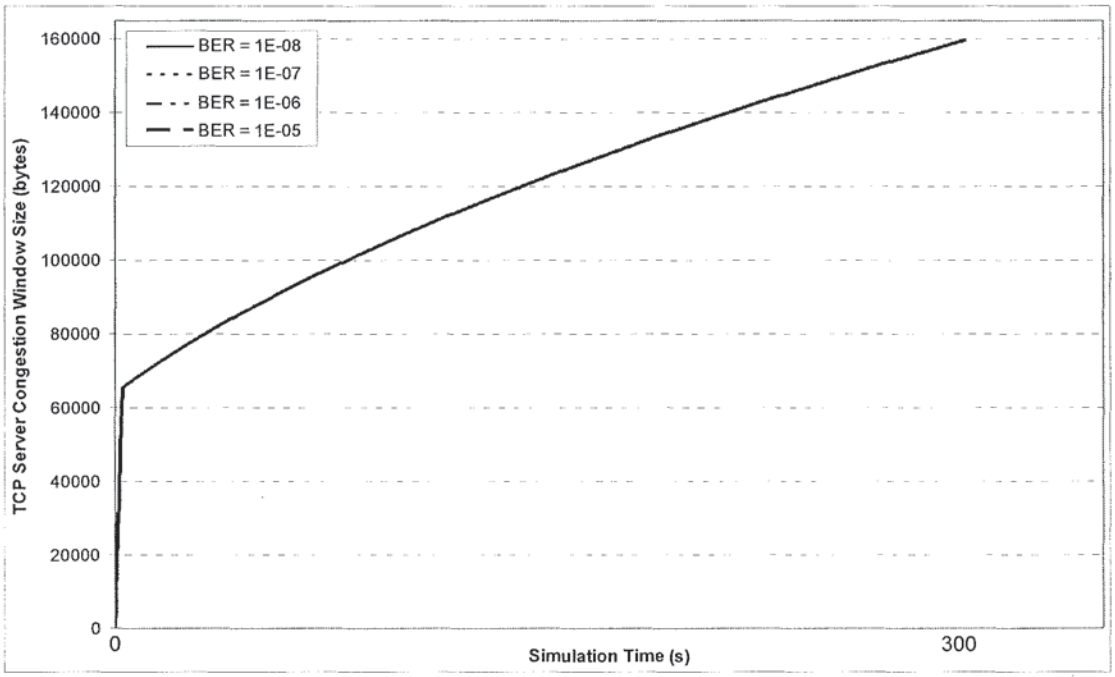


Figure 4.5: The *cwnd* evolution of a TCP sender for wired-to-wireless paths under varying BERs

Figure 4.6 is a plot of the *cwnd* behaviour for a BER value of 10^{-4} over the WLAN. It is plotted separately because it represented a turning point in the results. It can be seen from Figure 4.6 that the *cwnd* possesses very erratic behaviour. Due to the severe volatility and repetitiveness of the curve behaviour throughout the entire simulation, only a subset of the full time sequence has been plotted for legibility purposes, as the objective here is to purely highlight the behaviour of the *cwnd* to the reader. As is clearly visible from Figure 4.6, the sender's *cwnd* never enters the congestion avoidance phase. It is able to initiate the slow start phase, but before it gets the chance to grow to the *ssthresh* size it drops down to the size of a single segment. This is due to the wireless channel possessing a BER that is significantly high, causing a greater number of unrecoverable segment losses that cannot be retrieved even by local 802.11 MAC retransmissions. This causes many DUPACKs to arrive at the TCP sender that is forced to perform a fast retransmission of the missing segments, dropping its *cwnd* size

each time to one segment size due to the fact that it is still in the slow start phase. Constant losses cause constant fast retransmits at the sender, and this is reflected in Figure 4.6, showing that the *cwnd* never grows beyond 11 Kb (which was the maximum for the entire simulation). In contrast, the *cwnd* in Figure 4.5 for the lower BER conditions manages to grow to a size of 160 Kb, which is more than eleven times the size.

When subjecting the network to BER conditions in the order of 10^{-3} simulations results were unobtainable as the error conditions were simply too severe for TCP to function appropriately. Hence, all remaining plots in this section do not include BER values of this magnitude.

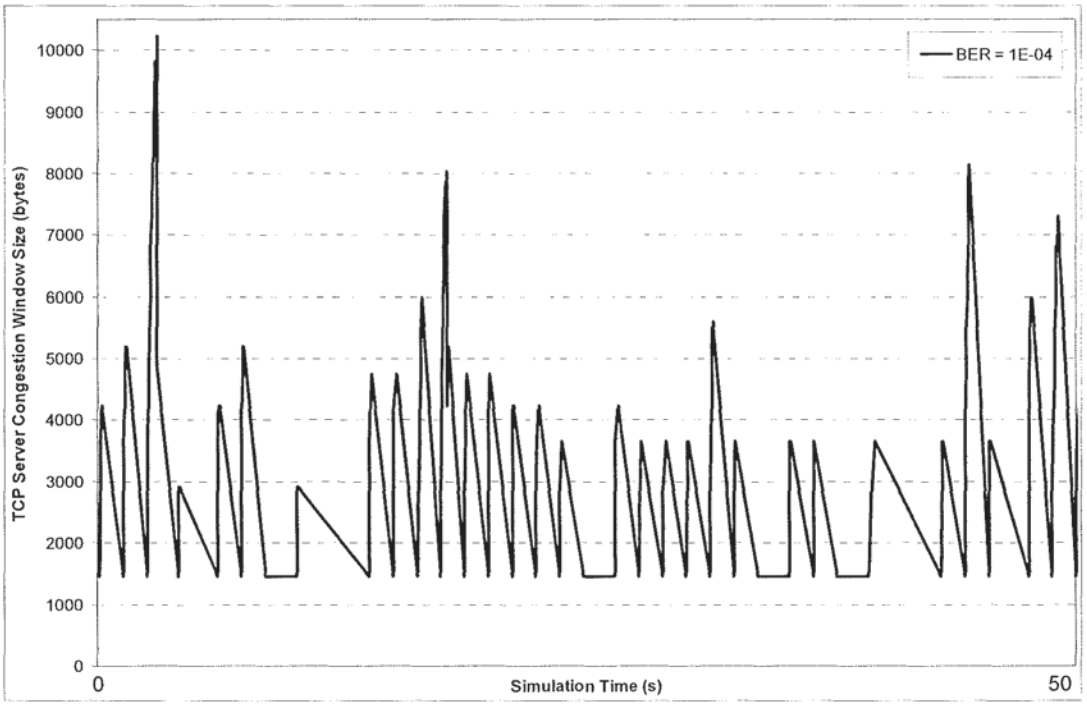


Figure 4.6: The *cwnd* evolution of a TCP sender for a wired-to-wireless path with a high BER

In Figure 4.7, the average end-to-end RTT and RTO timer values at the TCP sender have been plotted for the entire range of BER values tested (note that this is not the application-layer delay). An interesting observation from this plot is that the TCP RTO timer value is more sensitive to increasing BERs over the WLAN than the actual RTT. This significant increase can be explained by consecutive timeouts caused by retransmitted segments that are also lost, which causes an exponential increase in the value of the timer for each unsuccessful attempt. A high RTO timer value will cause

long delays between retransmissions at the server, potentially stalling the connection for several seconds between attempts on average.

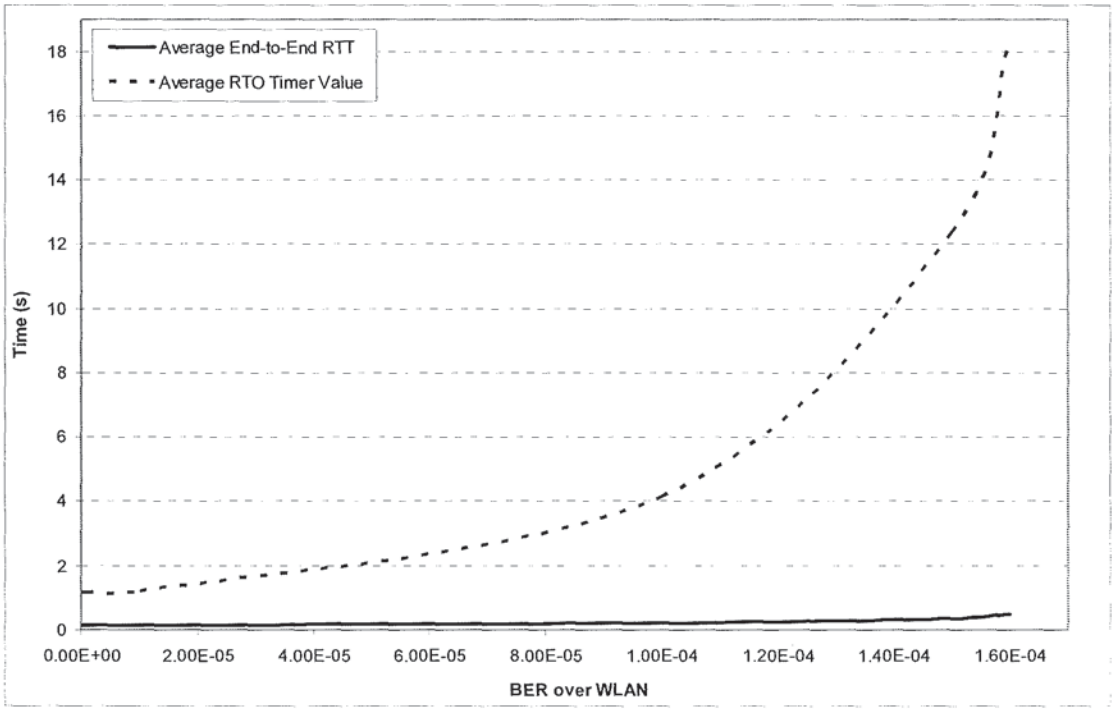


Figure 4.7: Average TCP RTT and RTO timer values per connection under varying BERs

In Figure 4.8, the retransmission behaviour of the ARQ mechanism of the 802.11 MAC at the AP has been plotted for the same simulations presented in Figure 4.7. It presents the trend for the average number of retransmission attempts made for each 802.11 data frame transmission towards the end-device (where each frame encapsulated a TCP data segment). Note that at very low BERs over the WLAN, the average number of retransmissions made is zero, which is expected. As the BER value increases, so does the average number of retransmissions. A higher number of retransmission per frame leads to a TCP segment experiencing longer delays before it successfully reaches the receiver. This will increase the end-to-end RTT perceived by the TCP sender as it waits for an ACK to arrive. When the RTT increases, the sender's computation of the RTO timer will also increase. Unfortunately, such increases in the timer value can stall a TCP sender whilst waiting for ACKs to arrive, which eventually never will. The delay will only lead to a timeout event and retransmissions by the sender, with a severe reduction in the size of *cwnd*. It can be seen from Figure 4.8 that at a BER of 1.0E-04 the average number of ARQ attempts per frame at the AP was

2.5, which was significant enough to cause the *cwnd* behaviour seen in Figure 4.6. Therefore, the results reveal that the 802.11 MAC ARQ function is only able to protect a TCP sender from the lossy WLAN channel for BERs of less than 10^{-4}.

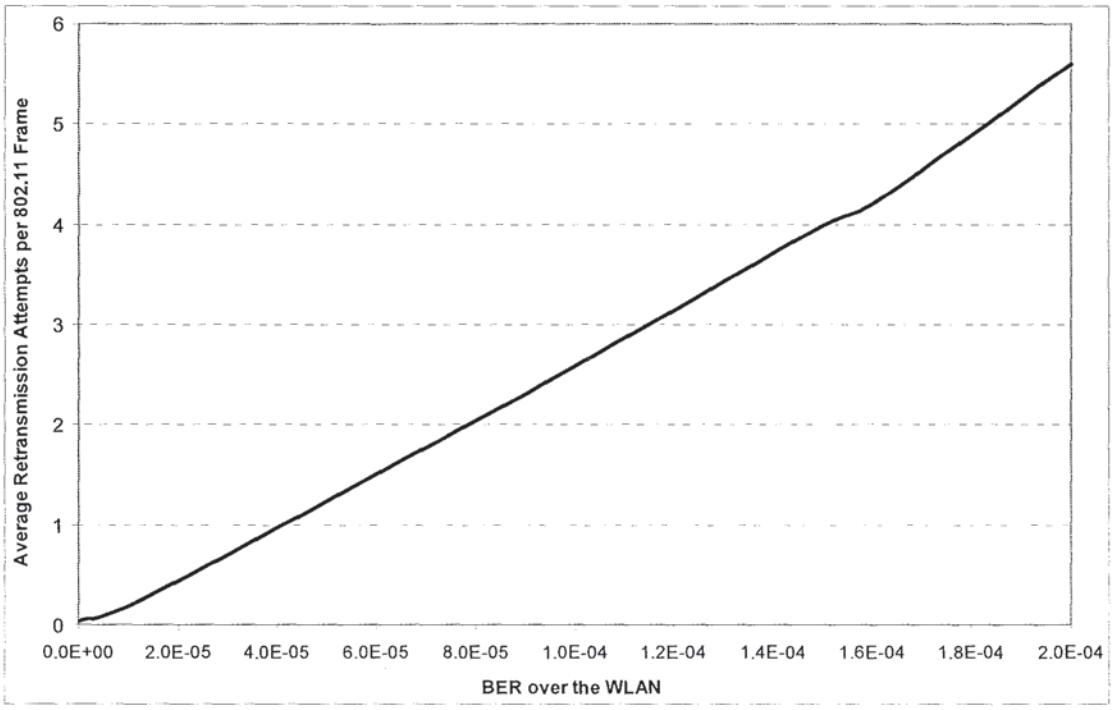


Figure 4.8: Average number of frame retransmission attempts made by 802.11 MAC at the AP

4.5.2 Throughput Performance over the WLAN

In Figure 4.9 the maximum achieved data throughput performance over the WLAN under varying BERs is presented, as perceived at the TCP layer on the 802.11 device. The results in Figure 4.9 correlate to those shown in Figures 4.6 and 4.7 from the same simulations. In Figure 4.9, it can be seen that the maximum achieved throughput was slightly higher than 5 Mbps in very good channel conditions (~10^{-8}) over the WLAN, which is significantly lower than the theoretical maximum of 11 Mbps as defined by the 802.11b standard as would be expected. The lower than expected throughput here is attributable to the processing overheads of the 802.11 MAC and PHY layers, rather than transmission errors, and is an entirely separate area of study for 802.11b/g performance [139]. The turning point in the results for the maximum throughput occurred at a BER of 3.0E-05 (~10^{-5}), where there was a sharp drop in the achievable throughput to less than 1 Mbps that occurred within a BER reduction of a single order of magnitude to 10^{-4}. From looking at Figure 4.9 it is fair to say that although the

turning point appears to be at a BER of ~10^{-5}, the true cut off point for 802.11 end-user performance is more likely to be at a BER of ~10^{-4}. Further increases in the BER caused the throughput to diminish to values that would otherwise render useless to end-users of TCP applications.

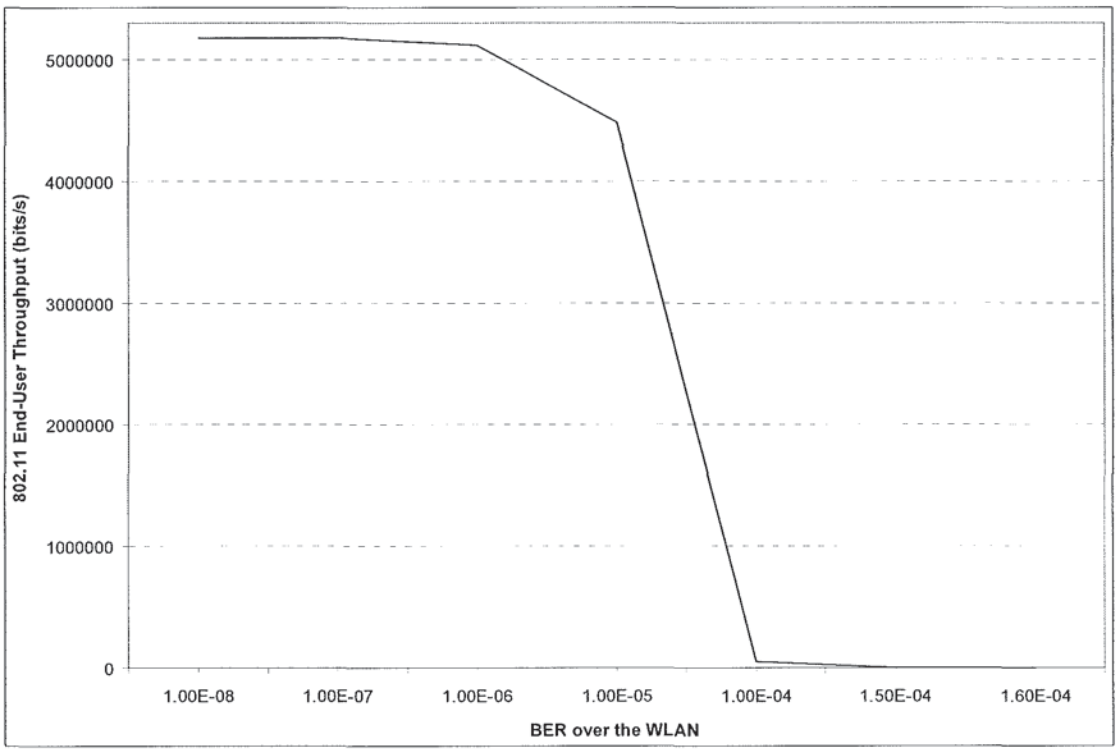


Figure 4.9: Maximum throughput performance over the 802.11 WLAN

4.5.3 Transfer Time Performance for FTP Downloads

In this section the focus is specifically on FTP performance of the 15 Mb file downloads by the 802.11 end-device. Figure 4.10 plots the total time taken to complete the entire transfer (from the TCP server) over varying BER conditions over the WLAN, as perceived by the FTP client on the receiving device (which sits above the TCP layer). It can be seen that for BERs of less than around 10^{-6} the download times were quite acceptable, with the transfer completing in less than 60 seconds. Even at a slightly higher BER of 2E-05 the file transfer time increased six-fold significantly to 360 seconds (6 minutes), which could still be tolerated by certain end-users. For BERs between 3E-05 and 1E-04 the increases in download times continue at the same rate; however, for 802.11 end-users the download times would now be considered unacceptable, taking almost 47 minutes to complete at the latter BER value. The

critical point within the results was from BERs of $2x10^{-5}$ and higher, where a dramatic increase in the download times to complete the transfers is seen. It was established earlier that 10^{-4} was the turning point for the behaviour of the TCP sender's *cwnd* (Figure 4.6), and this is the most likely cause for the sharp drop in throughput due to a starved sending rate, leading to a sharp rise in transfer times. In fact, Figures 4.9 and 4.10 complement each other at BERs of $\sim 10^{-5}$.

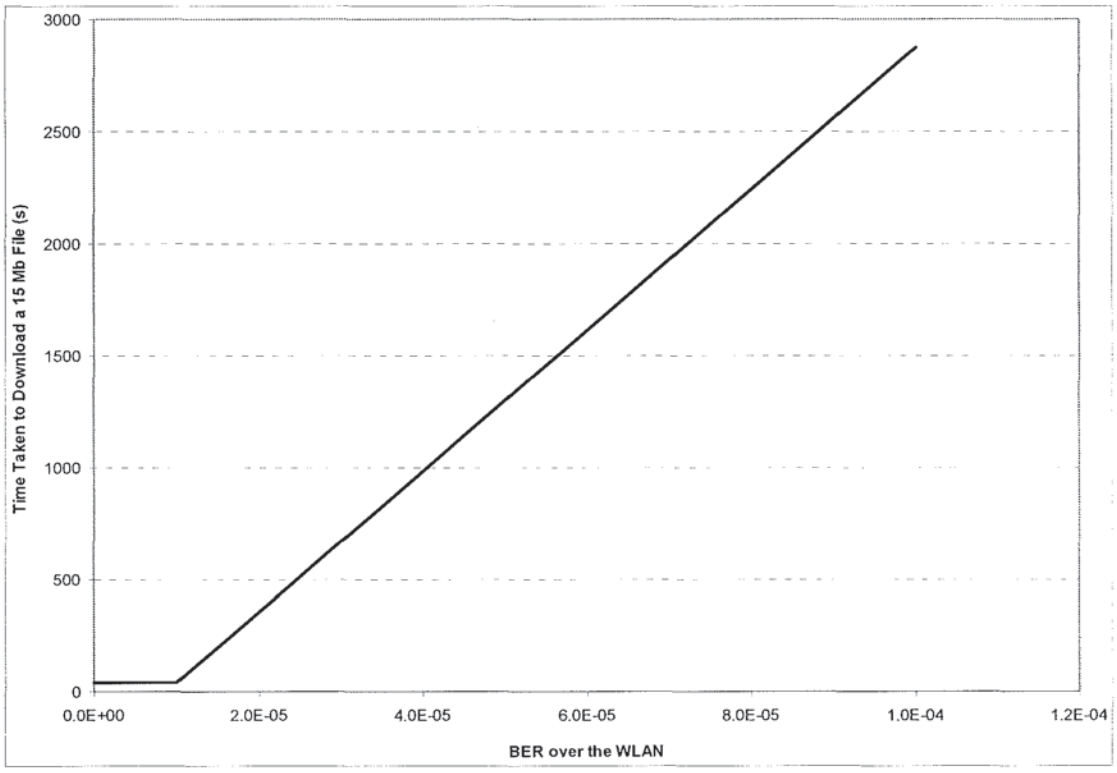


Figure 4.10: Time to download 15 Mb files by the 802.11 device using FTP under varying BERs

4.5.4 Web-page Response Times for HTTP Requests

Since HTTP web-page requests make up for a high proportion of TCP traffic in today's Internet [252], it was chosen as an interesting statistic to measure. In this section the focus is on end-users in the WLAN accessing web-pages from HTTP web-servers in the Internet, where TCP is used to reliably transport the data to the requesting end-users. All measurements were made on the receiver-side application layer at the HTTP client. Figure 4.11 plots the response times for HTTP 1.1 [253] web-page downloads, i.e. the time taken to completely download and display the contents of an entire web-page including all of its image objects. The results revealed some interesting characteristics. Up to BERs of around 1.5E-04 over the WLAN the web-

page download performance was still at acceptable levels for end-users, taking no more than 10 seconds per completed request. It took a BER of 2E-04 and higher to really degrade the download performance significantly as to render web-page browsing useless to the end-user device. The results indicate that HTTP traffic can sustain acceptable performance for higher BERs than FTP traffic. This could be explained by the fact that HTTP 1.1 clients normally make *pipelined requests* over single TCP connections, which means that a server is able to pack different HTTP objects into the least number of TCP segments to be transmitted. Since web-pages contain only small amounts of total data to start with, then there will be fewer TCP segments to send over the wireless path, which are also less likely to experience bit corruptions over the entire range of segments for a particular web-page.

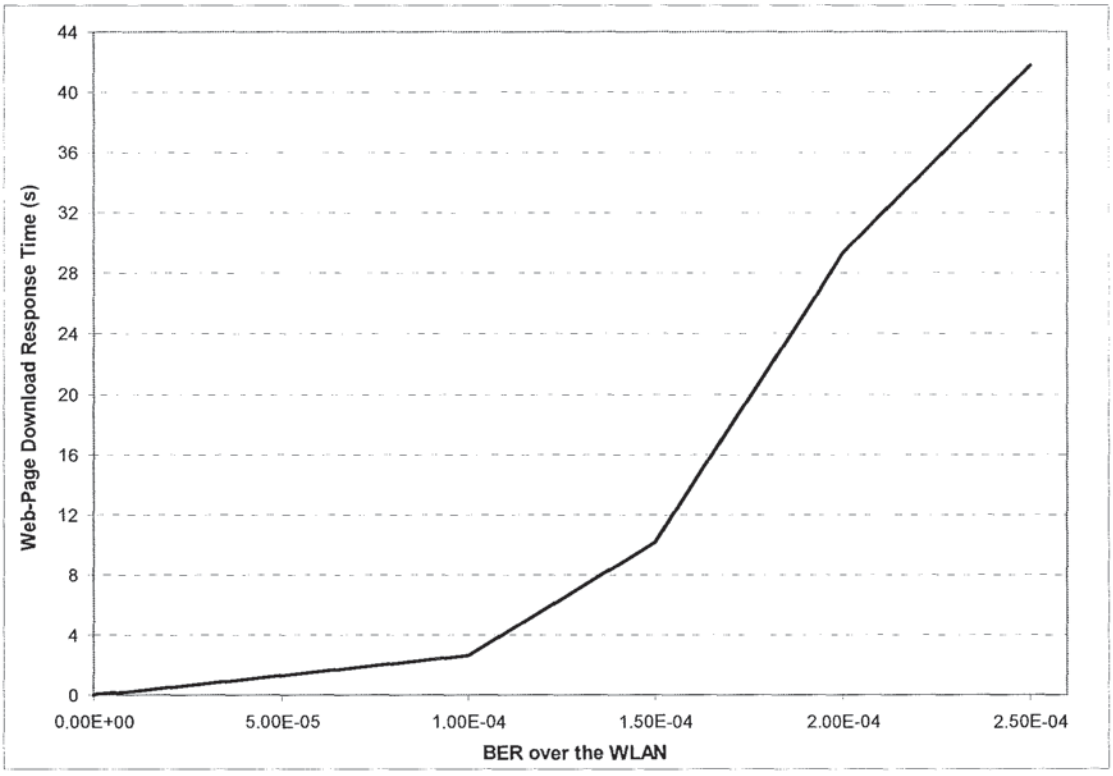


Figure 4.11: Web-page download times for 802.11 device using HTTP under varying BERs

4.6 Chapter Conclusions

In this chapter the performance of the widely used TCP Reno, a true end-to-end transport protocol, has been investigated when used over wired-to-wireless

environments. The focus has been on TCP connections running over 802.11 WLANs at the last-hop, under varying BERs over the radio link.

Simulation results using the comprehensive OPNET Modeler™ v14.0 have highlighted the ineffectiveness of TCP Reno in conditions where the channel BER is in excess of ~10^{-5}, affecting both end-users of popular Internet application-layer protocols such as HTTP and FTP. The results have also revealed that HTTP web-page traffic performance can sustain higher BERs over the wireless link than larger sized file downloads from the Internet using FTP.

Chapter 5

Unidirectional vs. Bidirectional Losses

5.1 Introduction

The continued popularity of mobile and wireless networking has diverted the attention of many to the maximisation of performance of such technologies using legacy protocols that still govern the Internet. Hence much research is still being carried out with evaluating the performance of TCP over wireless channels with varying conditions, with many resorting to analytical models in recent years to predict sender-side behaviour [31] [55] [254] [255]. As discovered in the preceding chapter, the key issue for wired TCP senders over wireless paths is that of higher packet loss rates of a random nature due to higher BERs over the transmission channels, which were not accounted for in TCP's original design.

One of the challenges in gauging the accurate behaviour of TCP senders over wireless channels is the difficulty in setting up a testing platform for performing realistic experiments, as well as gathering accurate statistics on TCP's true behaviour at the sender side, which is usually a server machine located somewhere in the Internet. Another challenge is in the acquisition of correct equipment and tools in order to set up a real-world wireless testbed that allows detailed measurements of TCP traffic to be made. Hence, researchers often resort to simulation, emulation, and analytical modelling as ways of experimenting with TCP, using best-fit wireless channel models to replicate the real-world conditions as much as possible.

After much literature review in the area [140] [169] [154] [245] [256] [257] [258], it was discovered that researchers often make many assumptions about the conditions in which they are testing TCP. The key observation is that researchers were only very rarely making references to the error model being used on the wireless feedback (reverse) channel of experiments. In other words, all experiments were being conducted with emphasis on TCP segment losses occuring only on the forward path of

TCP connections. It is fair to say that this is not how it occurs in the real-world, where the reverse path is equally as erroneous as the forward path. In other words, a wireless channel possesses a particular BER, and all TCP connections that traverse this channel in either direction will experience losses in both the forward data path and the reverse ACK path, at the same BER [130]. It can therefore be inferred that researchers are potentially overestimating the performance of TCP from their results and evaluations.

The problem with TCP and the feedback channel is relatively simple. TCP is a reliable transport protocol and uses the concept of acknowledging all data that is transmitted (or received) successfully [49]. When a TCP sender transmits a data segment in the forward direction to a wireless receiver, upon its successful arrival at the TCP layer the receiver transmits an ACK back to the sender over the reverse wireless channel to inform of its arrival, i.e. providing feedback. Since the behaviour (and hence performance) of TCP is heavily based on this feedback arriving back at the sender in a regular manner, it becomes an issue for TCP when ACKs are lost over wireless channels on the reverse path. In other words a TCP sender relies on returning ACKs in order to advance its *cwnd* so that the amount of data transmitted in a single flight is always increasing up to a maximum capacity, which directly affects throughput performance. However, by completely ignoring ACK losses on the feedback channel researchers are potentially assuming and experimenting in unrealistic conditions.

The motivation behind the work in this chapter is therefore to clearly highlight the differences in performance that are produced when experiments are performed with and without ACK losses on the feedback channel of a TCP connection. It is hoped that the results will encourage researchers of TCP over wireless networks to re-assess their channel models before conducting performance experiments, which can only lead to more robust TCP implementations that can survive in the real-world too.

5.2 Related Work and Motivations

There are only a few publications discovered that boldly highlight the differences between unidirectional and bidirectional error conditions for TCP connections in simulation and emulation environments, something which will be emphasised in this chapter.

In [128], the authors devote much of their time to investigating the significance of asymmetry between the forward and reverse channels on TCP performance, however the emphasis of their work is mainly on the asymmetry related to network bandwidth and latencies introduced as a result.

In [259], the authors have devoted much effort to studying the impacts of the reverse channel on TCP ACKs, in particular to the idea of *ACK filtering* to overcome the effects of slower reverse paths for TCP connections. Again, their work does not deal directly with losses associated with wireless error conditions, and little reference has been made to performance differences as a result of this.

In [260] and [261], the authors also experiment with asymmetrical channels, but the emphasis of the work is on the bottleneck link that usually occurs on the reverse path for TCP connections, delaying ACKs as opposed to causing losses. The authors in both papers do however acknowledge the heavy reliance of TCP on returning ACKs, and also highlight the significant impacts on TCP senders when ACKs are delayed or do not arrive at all.

In [262] the authors have undertaken a comparison and performance evaluation of various sender-side TCP algorithms with the focus specifically on reverse channel traffic characteristics. The authors highlight their motivations by also pointing out several previous studies where the reverse TCP channel traffic has been neglected, arguing that feedback traffic (i.e. ACKs) has a significant influence on the end-to-end performance of TCP. However the focus of their work is on the asymmetry between upstream and downstream TCP traffic due to heavy congestion, where ACKs are lost and/or delayed due to excessive queuing along routers. They go on to perform simulations to highlight the impacts of such effects on several TCP implementations to support their claims. Although the work in this paper is indicative of TCP's dependence on reverse channel conditions, the authors do not consider ACK losses that occur randomly due to a wireless link. Their simulation studies do not use a controlled loss rate variable for the reverse path; rather this is done through initiating several TCP connections over a bottleneck link.

5.3 Experimental Setup and Procedures

In this section the emulation environment for studying and comparing the impacts of unidirectional versus bidirectional packet losses on end-to-end TCP performance is presented, followed by the measurement tools and techniques that were used.

5.3.1 End-to-End TCP Emulation Platform

All experiments were conducted over an emulation platform using the popular *netem* [263] tool. Netem provides network emulation functionalities for testing higher-layer protocols by reproducing the characteristics of real networks on traffic, such as variable delays, packet losses, duplications, and reordering. Essentially netem operates as a queuing discipline within the Linux kernel network stack.

A simple server-client topology was built using real high-end *Pentium 4* machines, each with 1024 Mbytes of system memory. The platform was configured to emulate conditions that would occur in end-to-end TCP connections between a server (sender) located in the Internet and a last-hop wireless client (receiver). Figure 5.1 illustrates the end-to-end emulation topology that was used.

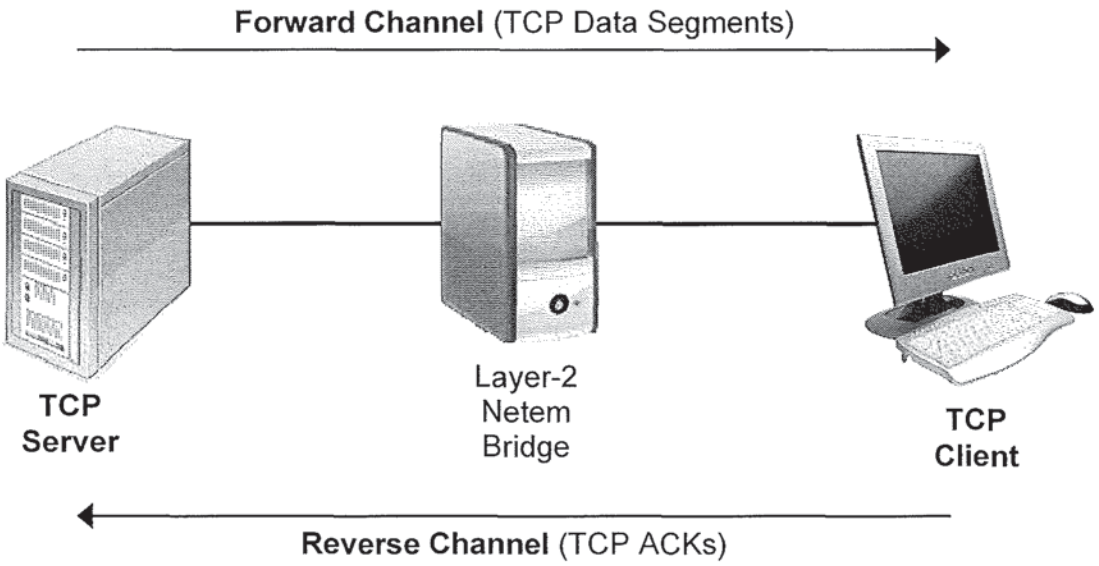


Figure 5.1: The TCP emulation platform topology

The sender and receiver machines were both based on the Linux operating system, each running version 2.6.18 of the kernel. The sending machine's kernel was also 'instrumented' in order to extract live TCP variables during experiments, such as the *cwnd* values, and so forth.

To emulate the effects of a lossy wireless channel with a range of adequate random packet loss rates, netem was installed on a separate Linux machine configured as a network *Linux bridge* [264] between the sender and receiver machines via cross-over 10/100 Mbps Ethernet cables to eliminate the need for using additional components. A Linux bridge allows two physical Ethernet *network interface cards* (NICs) to be 'bridged' at Layer-2 of the OSI stack (i.e. at the Ethernet layer). The bridging of two NICs allows the forwarding of all higher-layer traffic from one NIC to the other, in both directions. A bridge therefore transparently relays traffic between two NICs. The bridge has no awareness of the protocols it forwards, it only sees *Ethernet frames*. As such, the bridging functionality is protocol independent. The use of netem on a separate bridge machine with two NICs is the ideal method of performing experiments using the random packet loss rate variable [263].

The Linux bridge machine was a high-end Pentium machine with 1024 Mb of system memory, hence the traffic forwarding delays over the bridge interface were negligible [265].

Using this particular setup (as shown in Figure 5.1), Ethernet frames can pass through the bridge in both directions transparently to the TCP sender and receiver on either side, be it data segments or ACKs. Netem was used to induce a random packet loss rate (as a percentage of frames to drop) to all traffic flowing through the bridge's network interfaces in both directions independently.

In order to highlight purely the differences in TCP performance in the presence of unidirectional packet losses versus bidirectional packet losses over the forward and reverse channels, to emulate a wireless channel, congestion-related packet losses were not introduced. It is also appreciated the fact that although random packet losses are artificially induced within the experiments, the use of real machines and protocols in the emulation platform still has credibility because it factors in the effects of media

access contention, protocol processing delays, hardware delays, and media propagation delays, which are all difficult to model realistically through simulations.

5.3.2 Traffic Generation and Chosen Variables

To minimise the effects of queueing delays and interactions between multiple TCP connections, only single TCP flows were generated using the popular *iperf* bandwidth measurement tool [266]. The focus was on the sender's TCP layer, and the overall performance statistics reported by iperf. Each TCP experiment consisted of sending 100 Mbytes of data from the sender machine to the client machine, and measuring the time taken to perform the transfer, the maximum achieved channel bandwidth, and a snapshot of the *cwnd* behaviour on the sending machine.

In the Linux 2.6.18 kernel there are a range of pluggable TCP congestion algorithms that can be selected at runtime [267]. Based on research findings, three different TCP algorithms were selected for testing against unidirectional and bidirectional packet loss conditions; *TCP BIC* [268], TCP Reno, and TCP Veno, the latter two of which have been discussed in a previous chapter.

TCP BIC is an optimised sender-side congestion control algorithm for high-speed networks with high latencies. It has been designed to meet the demands of today's fast long distance networks, with the core objective being to take advantage of unused end-to-end bandwidth, a problem that is exhibited by the legacy TCP algorithms due to their poor AIMD response times. The main feature of BIC is its unique *cwnd* growth function.

BIC was chosen for testing because it is currently the default TCP algorithm that is enabled upon compilation of the Linux v2.6.18 kernel, and hence it is most likely to be widely used in recent Linux senders in the Internet. Reno was chosen because it represents a legacy sender-side TCP implementation that is still widely used in the Internet today, setting some baseline standards. Veno was chosen because it is a recent sender-side proposal for being able to deal with random segment losses better than existing TCP algorithms, and the aim was to simply observe its respective authors' claims.

Both the sending machine and receiving machine were enabled with the SACK option, as this is typical of most implementations today, being able to deal with multiple TCP segment losses per RTT. The window scaling option was also used at both sides. A TCP MSS of 1448 bytes was used for all experiments, with a MTU of 1500 bytes.

Based on the findings of [123], random packet loss rates ranging from 10^{-6} (0.0001%) up to $7x10^{-2}$ (7%) were used at the bridge via netem, which are common to wireless channels. At packet loss rates greater than 7% the performance of TCP was too degraded to continue with experiments. The maximum network capacity was 100 Mbps, as governed by the Ethernet transmission medium used. The latency within the network was unaltered, as it was not significant in what was trying to be shown.

5.4 Results

We begin this section by presenting a range of results from various TCP experiments over the emulation platform. Discussions of the results are made in the following section.

5.4.1 Maximum End-to-End Bandwidth Achieved

The first metric presented is the maximum possible bandwidth (or channel capacity) that was achieved over the 100 Mbps forward and reverse paths between the TCP sender and the TCP receiver. As was described earlier, iperf was used in server-client mode to transfer a total of 100 Mbytes of data in the server-to-client direction. Hence, the reverse channel consisted of only ACKs generated by the receiver on their way to the sender. The tests for each TCP variant were repeated three times for each particular packet loss rate over the channel and the results in this section are averages of three runs.

Random packet loss rates as percentages were applied initially to the forward channel traffic only passing through the bridge machine (will be referred to as unidirectional errors). The experiments were then repeated with identical packet loss rates affecting both the forward and reverse channel traffic (will be referred to as bidirectional errors).

Direct comparisons could then be made to highlight the effects on the different TCP algorithms when a reverse channel is erroneous causing ACKs to be lost too.

Figure 5.2 presents the maximum achieved channel bandwidth performance of TCP Reno in unidirectional versus bidirectional error conditions, for random packet loss rates ranging from 0.0001% up to 6%. Likewise, Figure 5.3 compares the performance of TCP BIC, and Figure 5.4 shows TCP Veno's performance. Note that a packet loss rate of 2% is what would be seen typically in real-world wireless channels.

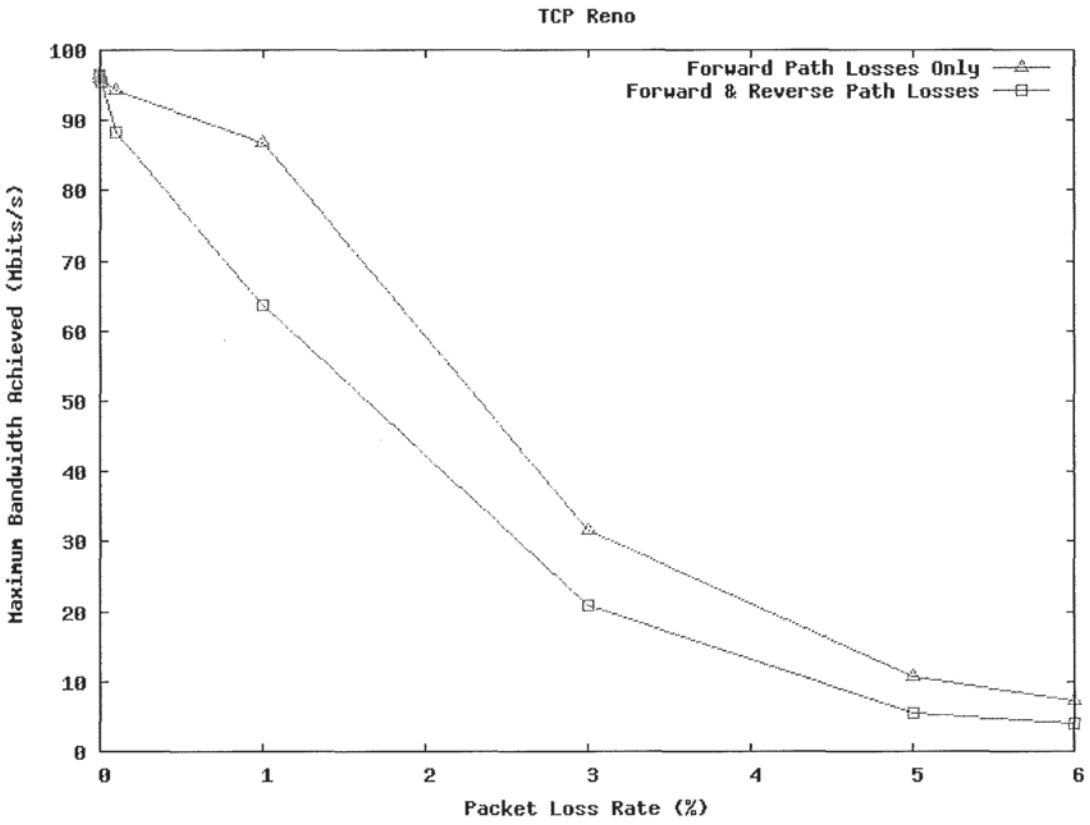


Figure 5.2: TCP Reno – Maximum achieved channel bandwidth performance

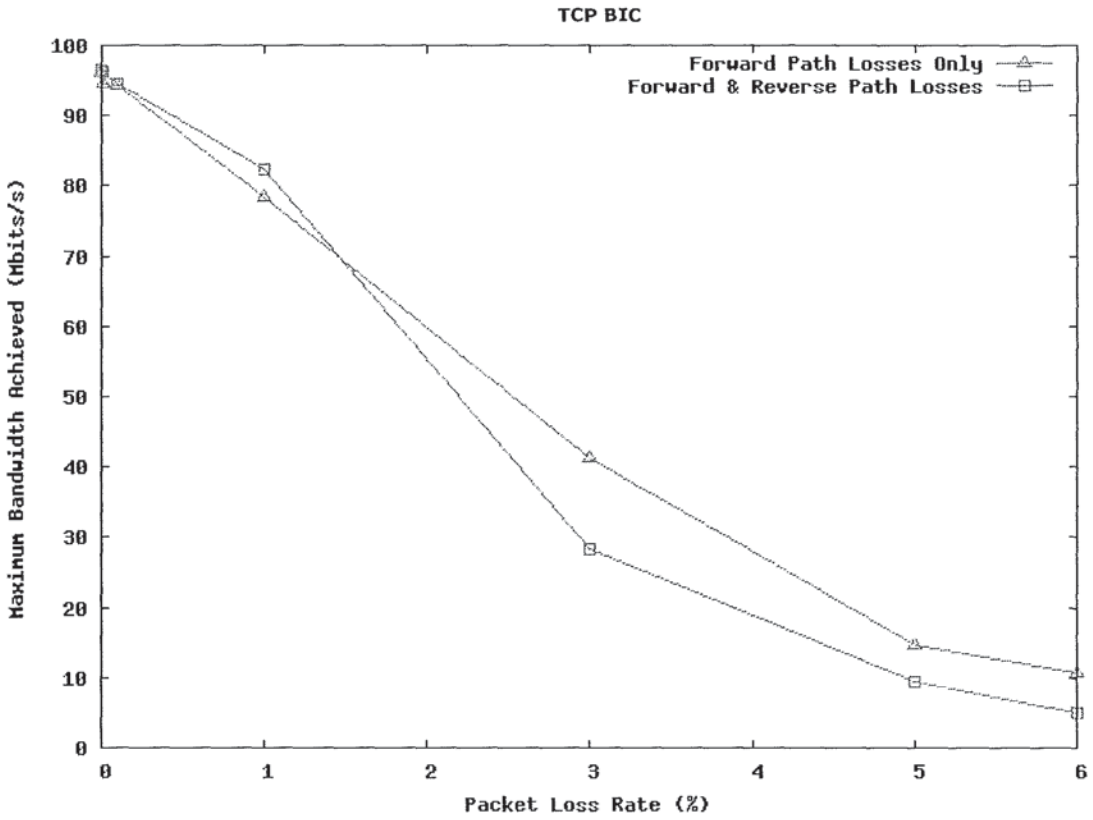


Figure 5.3: TCP BIC - Maximum achieved channel bandwidth performance

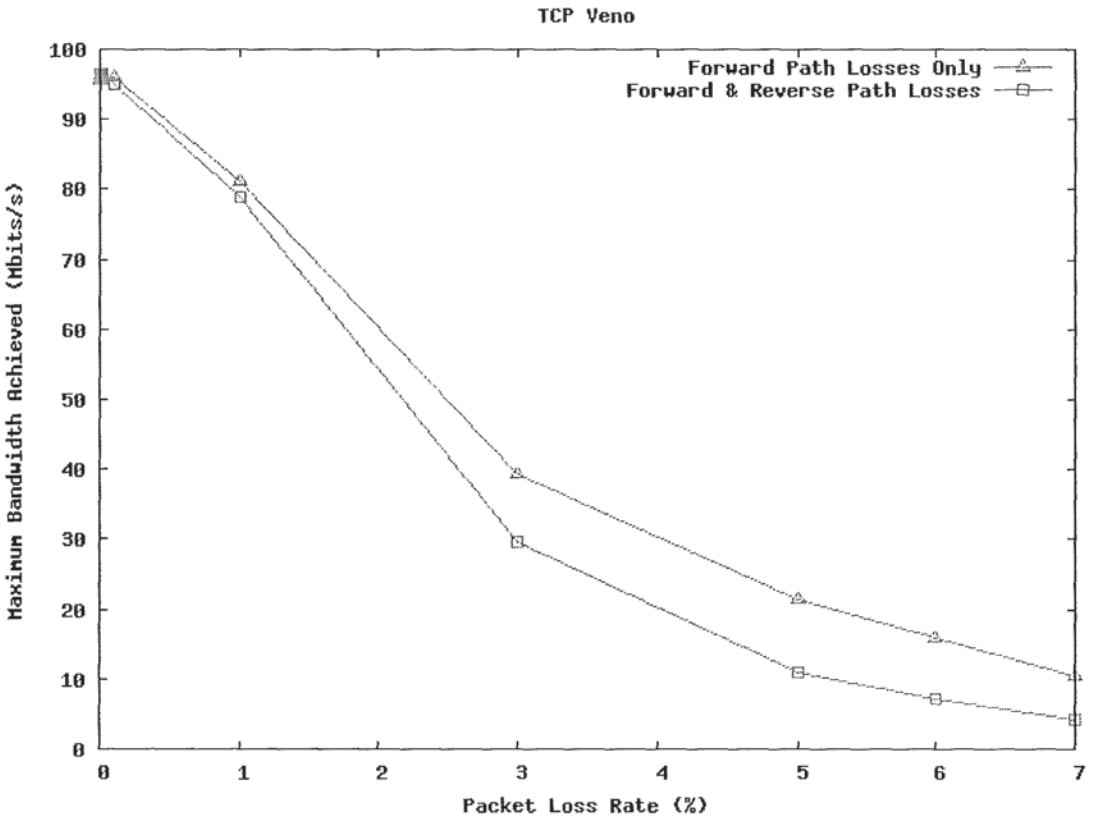


Figure 5.4: TCP Veno – Maximum achieved channel bandwidth performance

5.4.2 100 Mbytes Transfer Time Performance

In this section the second measurement taken for each of the experiments conducted are presented, i.e. the total time taken to transfer 100 Mb of data (as per Figures 5.2, 5.3, and 5.4). This metric was chosen because it is more meaningful, and easier to comprehend in terms of application-layer performance at the user level. Since iperf is persistent with the transferring of the data, it kept running until the entire 100 Mb was transferred, regardless of retransmissions and RTO events at the TCP sender, as well as other TCP features that would otherwise cause a connection to terminate such as connections which are idle for greater than a period of 3600 seconds. Again, average values have been presented in all plots.

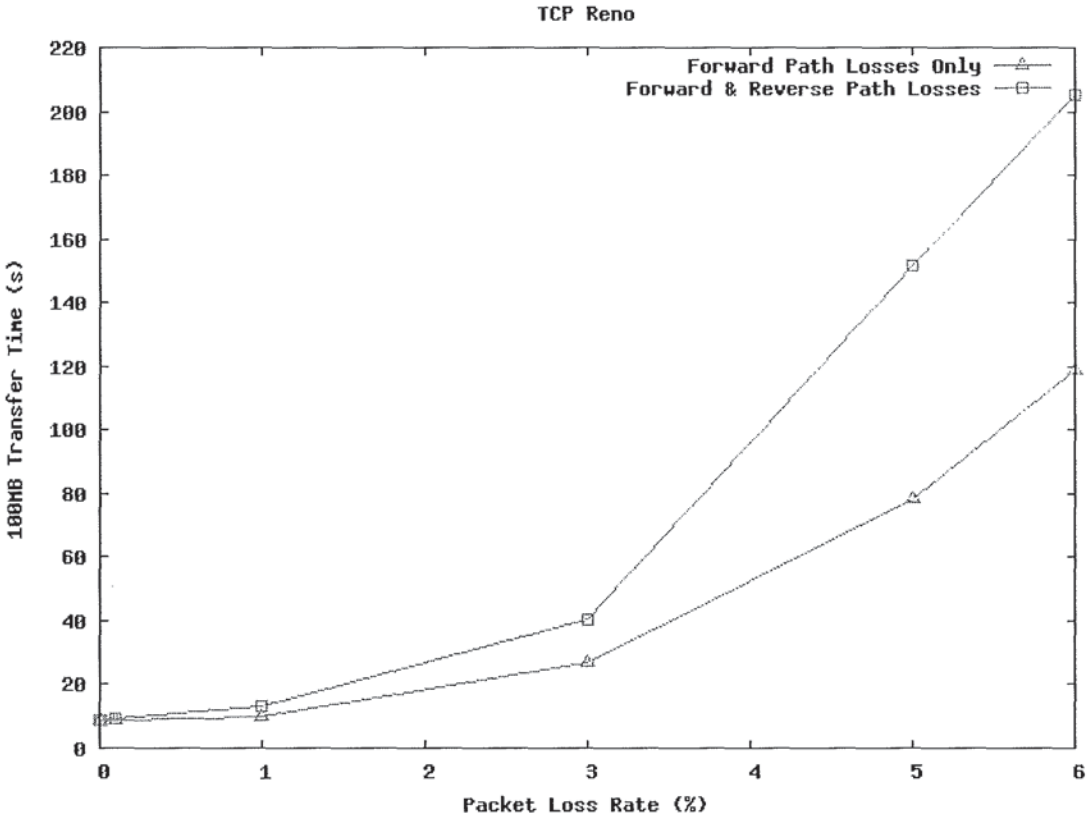


Figure 5.5: TCP Reno – 100 Mb transfer time performance

Figure 5.5 presents the performance of TCP Reno with regards to the total time taken to transfer 100 Mb of data (in the server-to-client direction) at varying channel packet loss rates, in both unidirectional and bidirectional scenarios. Figure 5.6 and Figure 5.7 represent the results for TCP BIC and TCP Veno, respectively. As a note to the reader, the results in Figures 5.5, 5.6, and 5.7 correspond to the same experiments as in Figures 5.2, 5.3, and 5.4, respectively.

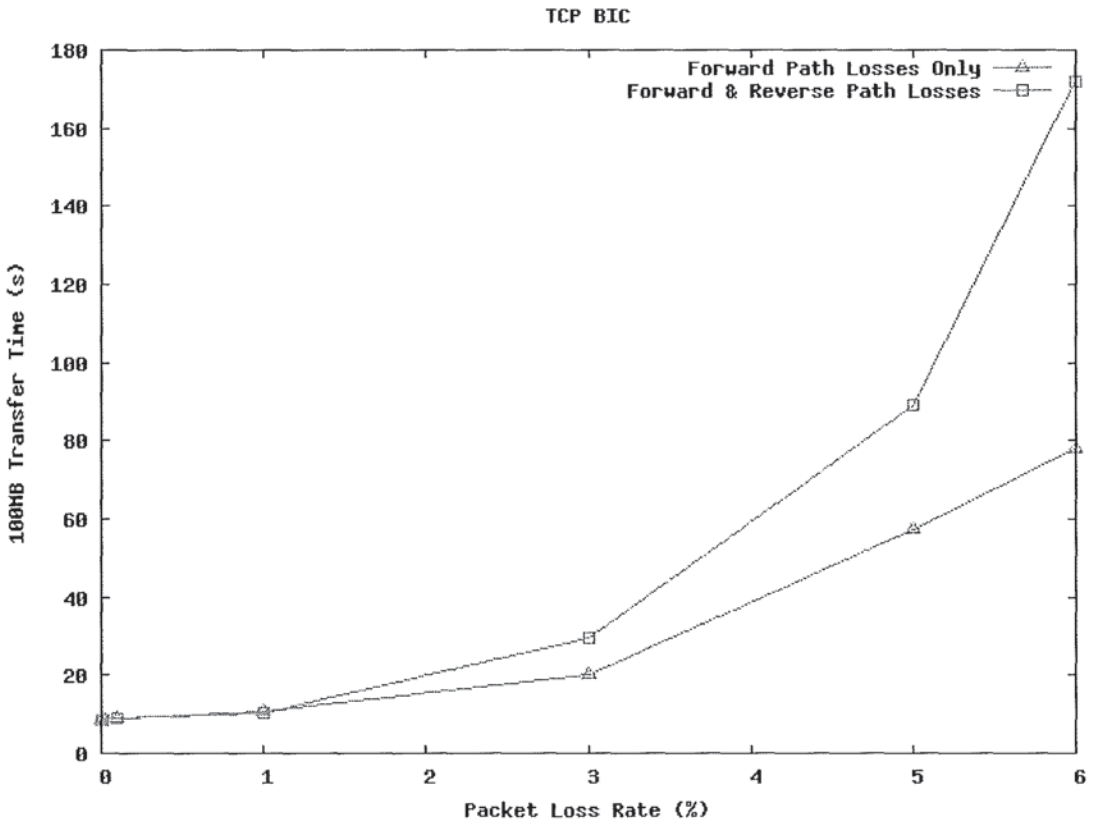


Figure 5.6: TCP BIC – 100 Mb transfer time performance

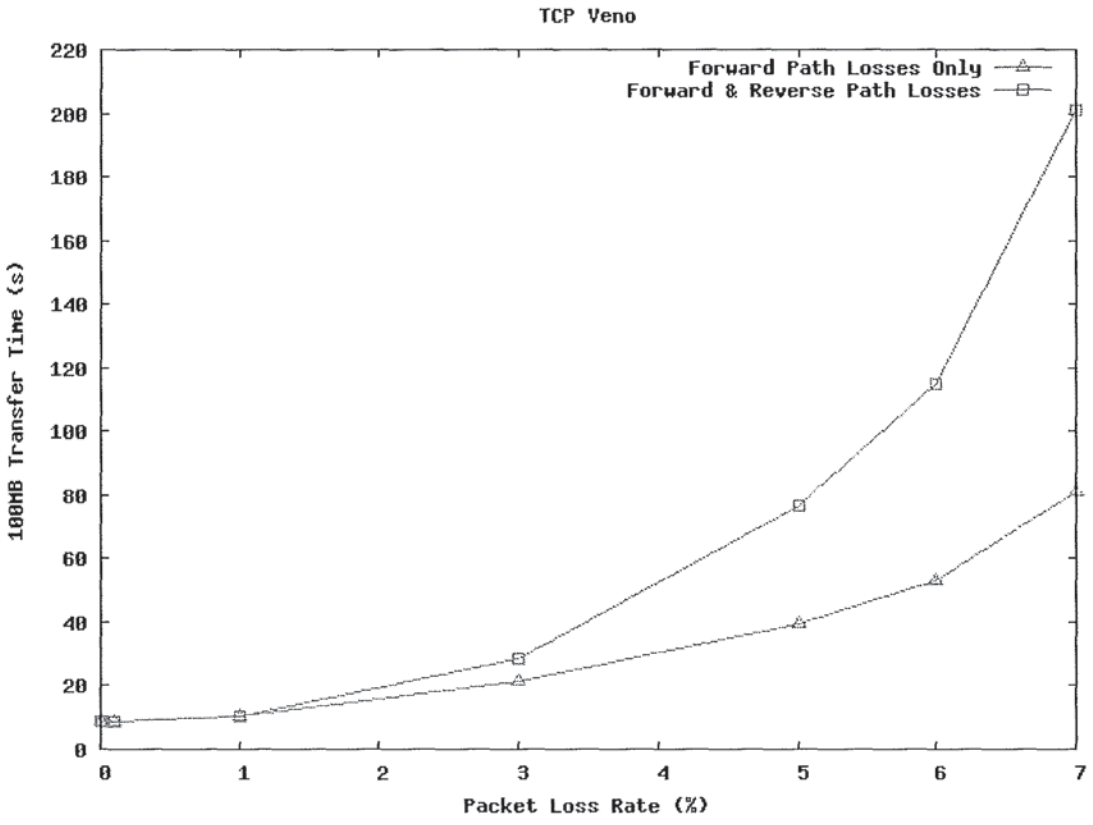


Figure 5.7: TCP Veno – 100 Mb transfer time performance

5.4.3 Sender Congestion Window Behaviour

In this section the impacts of bidirectional packet losses (i.e. ACK losses) on a TCP sender's *cwnd* behaviour are presented, supporting the general hypothesis of this chapter. Since the *cwnd* governs the overall sending performance of a TCP sender, observing its behaviour under varying random packet loss conditions can help to explain why performance is affected. All *cwnd* values were obtained from the TCP layer in the live Linux kernel during the experimental runs, and have been averaged over three runs per plot. Results for random packet loss rates of 0.001%, 0.01%, and 0.1% (applied at the bridge) have been presented for the Reno sender (Figures 5.8 to 5.10), the BIC sender (Figures 5.11 to 5.13), and the Veno sender (Figures 5.14 to 5.16), respectively. Higher packet loss rates produced very illegible plots due to the extremely erratic behaviour of the *cwnd*. Note that the plots in this section depict very accurate TCP sender *cwnd* behaviours of the real-world implementations in Linux v2.6.18.

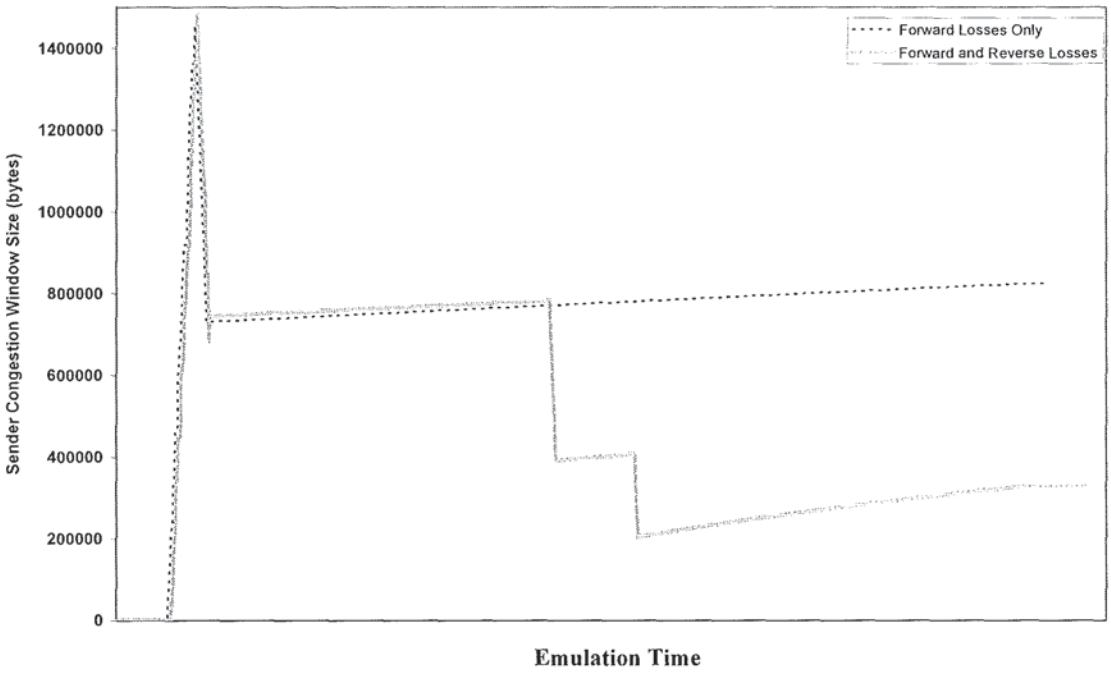


Figure 5.8: TCP Reno – 0.001% packet loss rate – unidirectional vs. bidirectional channel errors

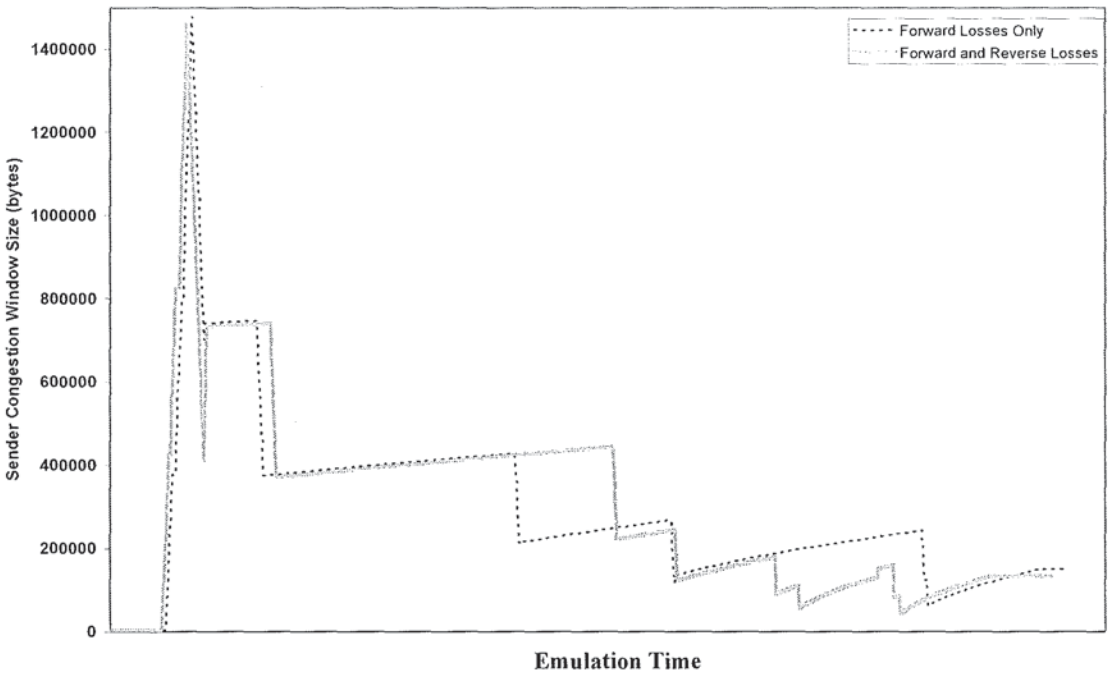


Figure 5.9: TCP Reno – 0.01% packet loss rate – unidirectional vs. bidirectional channel errors

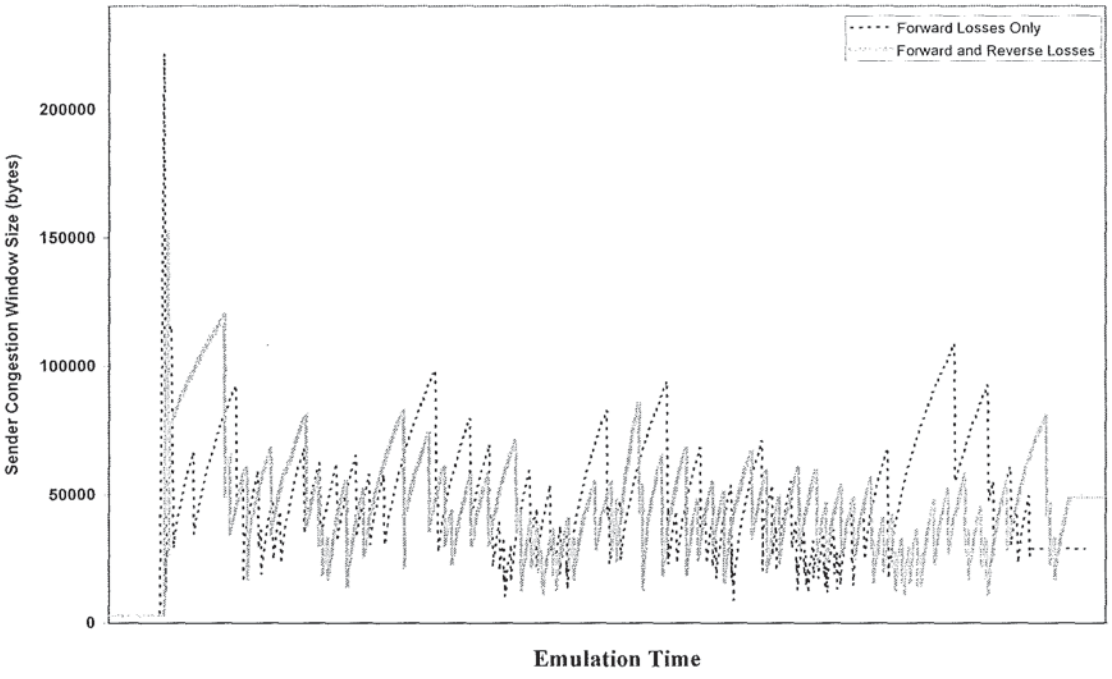


Figure 5.10: TCP Reno – 0.1% packet loss rate – unidirectional vs. bidirectional channel errors

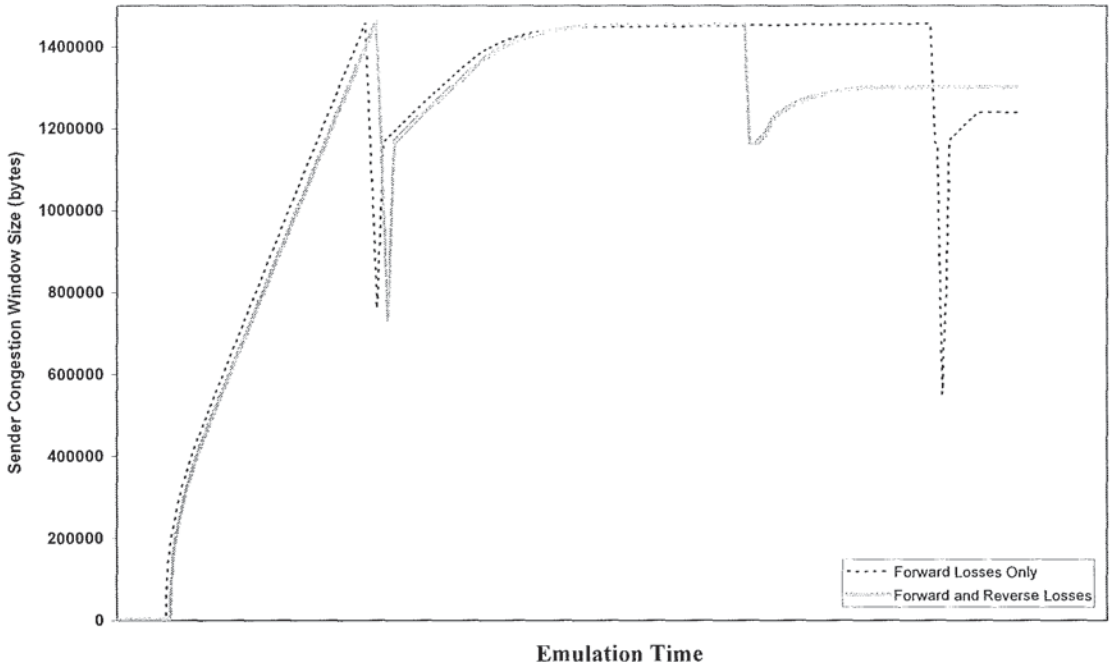


Figure 5.11: TCP BIC – 0.001% packet loss rate – unidirectional vs. bidirectional channel errors

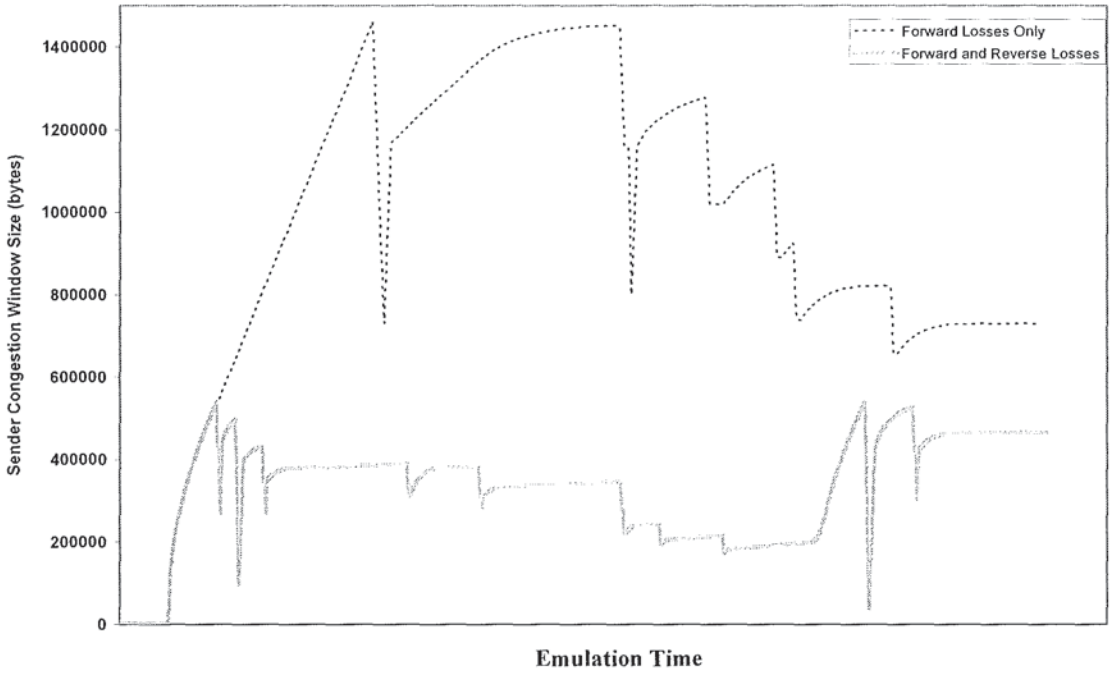


Figure 5.12: TCP BIC – 0.01% packet loss rate – unidirectional vs. bidirectional channel errors

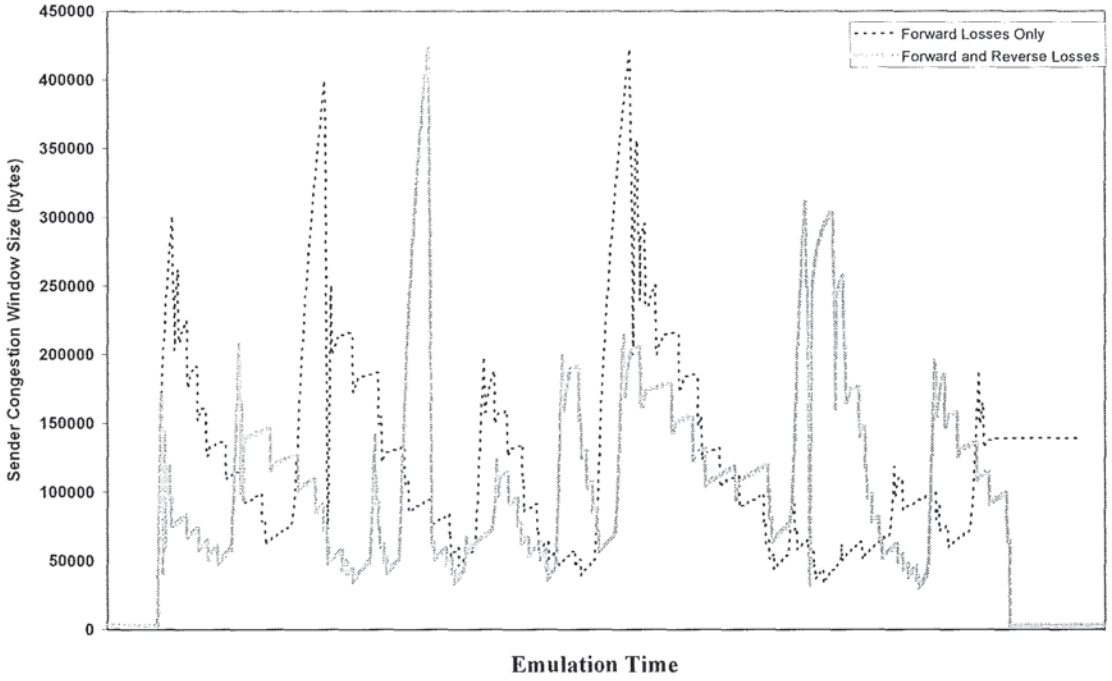


Figure 5.13: TCP BIC – 0.1% packet loss rate – unidirectional vs. bidirectional channel errors

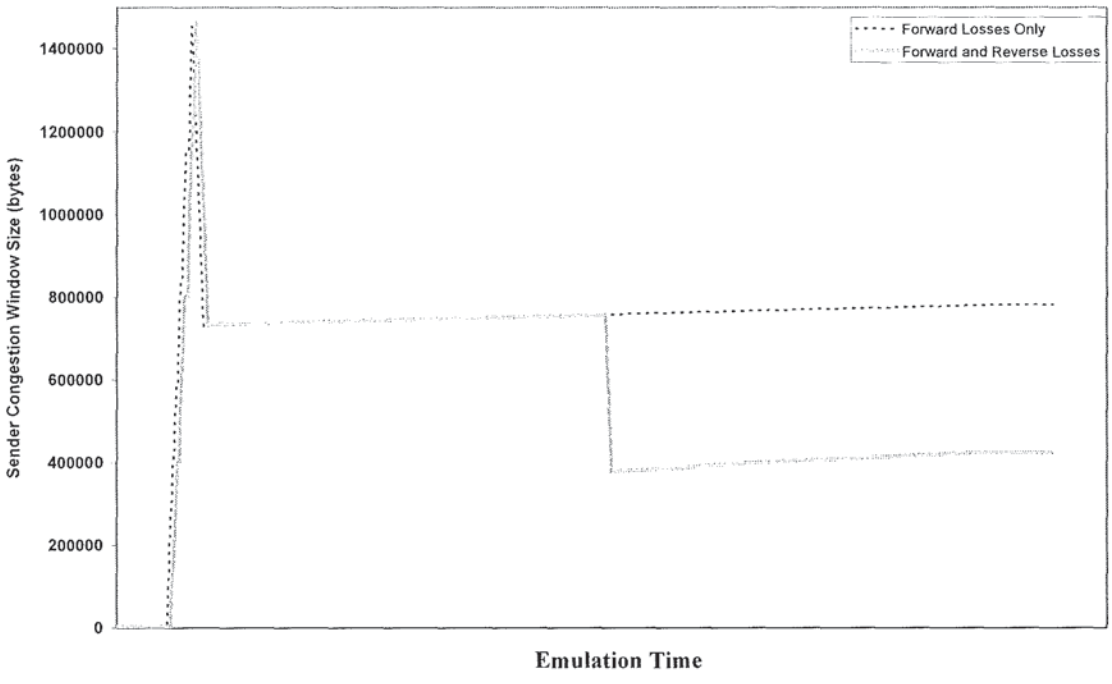


Figure 5.14: TCP Veno – 0.001% packet loss rate – unidirectional vs. bidirectional channel errors

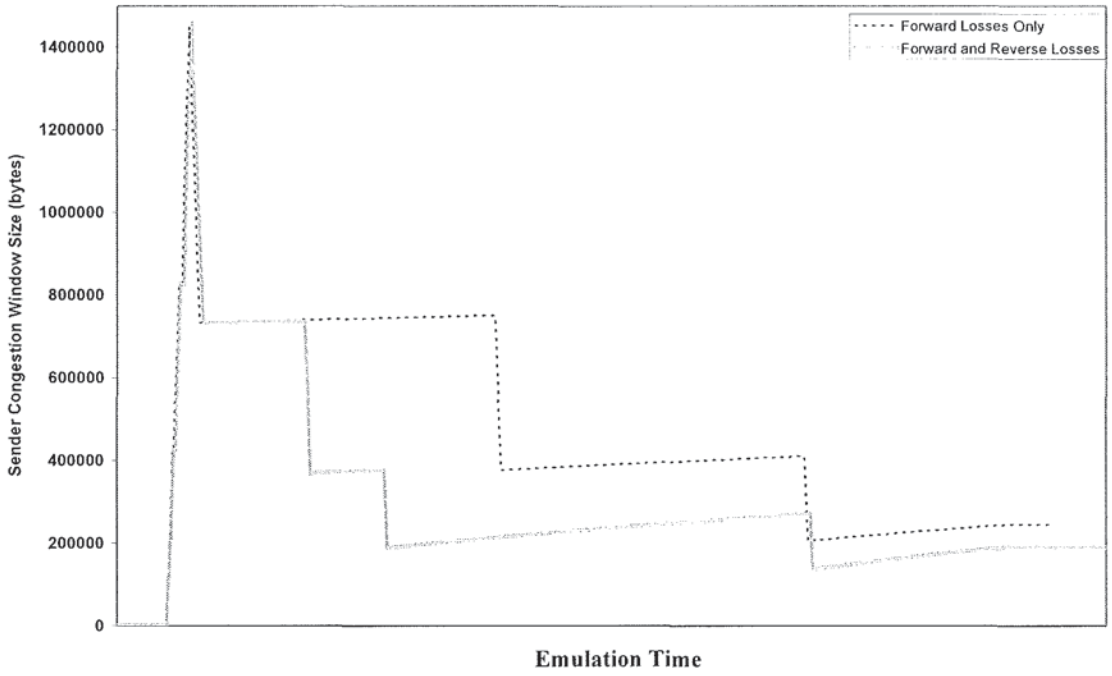


Figure 5.15: TCP Veno – 0.01% packet loss rate – unidirectional vs. bidirectional channel errors

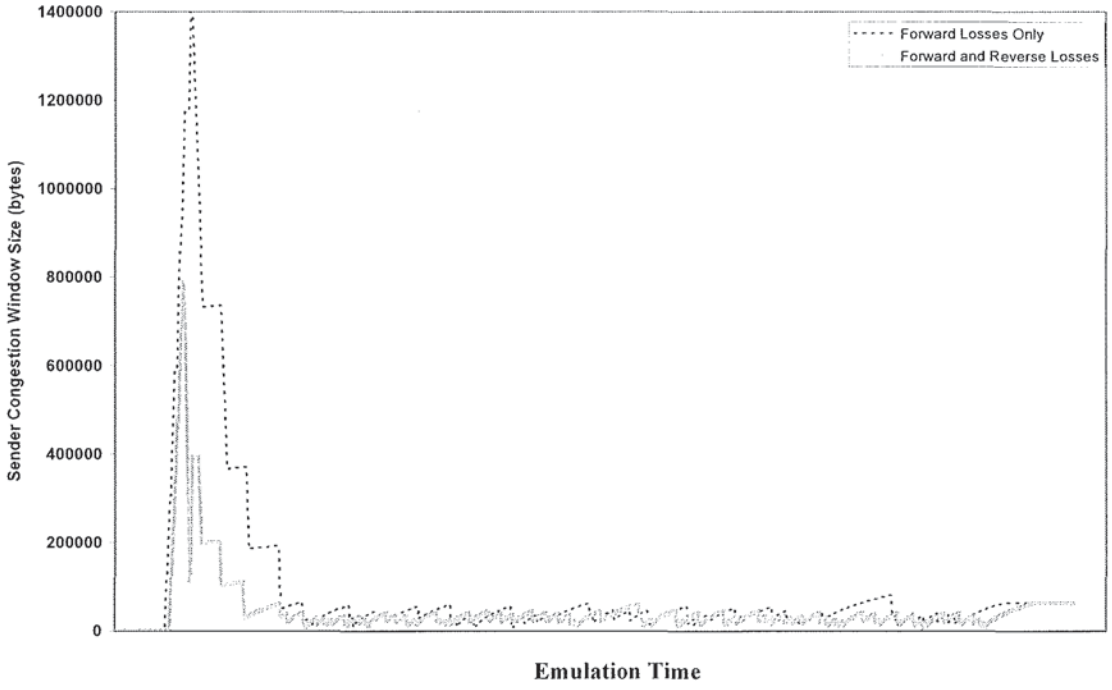


Figure 5.16: TCP Veno – 0.1% packet loss rate – unidirectional vs. bidirectional channel errors

5.5 Discussion of Results

It can be seen from Figures 5.2 (Reno), 5.3 (BIC), and 5.4 (Veno) that the maximum bandwidths achieved for TCP connections in forward-only (unidirectional) packet loss

conditions are consistently higher than for those connections where packet losses were introduced on the reverse channel too. This confirms the theory that when ACKs are subjected to lossy paths (such as over wireless channels), the evolution of the TCP sender's *cwnd* is affected, which then limits the amount of data that can be transmitted per RTT, i.e. the data rate at the sender is limited. The *cwnd* evolutions for each of the TCP algorithms under varying packet loss rates have been plotted for reference purposes in Figures 5.8 through to 5.16. Since the maximum bandwidth measurement by iperf is directly related to the maximum possible data rate over the channel by TCP, then as was expected, when the packet loss rate was increased for all experiments, the achieved channel bandwidth dropped as a result, although at different rates for the differing error scenarios and TCP variants. Ultimately this led to an increase in the times taken to send 100 Mb of as the packet loss rate increased, with unidirectional conditions producing better performance than bidirectional conditions across all TCP variants albeit at different rates. Figures 5.5 (Reno), 5.6 (BIC), and 5.7 (Veno) plot the transfer time performance under varying packet loss rates for both unidirectional and bidirectional conditions.

Figure 5.2 (Reno) shows that at a packet loss rate of 1% the unidirectional bandwidth measurement is 36% greater than for the equivalent bidirectional measurement. At packet loss rates of 3%, 5%, and 6% the corresponding differences in achieved bandwidth are 51%, 95%, and 73%, respectively. Figure 5.4 (Veno) is also consistent with this trend, showing similar significant differences in the achieved bandwidths between the unidirectional and bidirectional error conditions on the channel. In Figure 5.3 (BIC) it can be seen that at a packet loss rate of 3%, the difference between the unidirectional and bidirectional error conditions is quite significant; the achieved bandwidth for unidirectional packet losses is 46% greater than for bidirectional packet losses, and at a 5% packet loss rate the difference in achieved bandwidths is 55%. At a 6% packet loss rate the achieved bandwidth for unidirectional channel errors is more than twice that of what was achieved for bidirectional error conditions. These differences in performance highlight that an error-free reverse path for TCP experiments or simulations should not be assumed, as it will skew any results obtained. In reality, TCP ACKs do get lost over wireless channels.

Prior to progressing the discussions onto the results relating to the transfer times for 100 Mb of data, it is important to state what was considered to be an acceptable amount of time an end-user would be prepared to wait for the download of a 100 Mb file, for example. Taking the maximum capacity of the emulation platform into account (100 Mbps), a waiting time of up to 40 seconds was considered to be an acceptable level of performance.

First of all, looking at Figures 5.5 (Reno), 5.6 (BIC), and 5.7 (Veno), it is quite clear that the performance of all TCP algorithms in unidirectional error conditions is consistently better than when subjected to bidirectional errors conditions. To gain further insights into the magnitude of the differences between the two differing error scenarios, it was observed that the difference in percentages was very similar to those calculated for the maximum achieved bandwidth results. All three TCP variants were able to perform the 100 Mb transfer within 41 seconds for packet loss rates up to 3%, in both unidirectional and bidirectional conditions.

Further analysis led to an observation regarding the different variants of TCP in that the Veno algorithm performed better than its counterparts at the higher packet loss rates, for both unidirectional and bidirectional packet losses. Taking only the results for bidirectional packet losses, at a 5% packet loss rate Veno completed the 100 Mb transfer in 77 seconds. Comparing that to the likewise performance of Reno and BIC, their transfer times were 152 seconds and 89 seconds, respectively. At a 6% packet loss rate, Veno continues to outperform, taking 115 seconds for the transfer. At the same packet loss rate Reno took 205 seconds, and BIC took 172 seconds. It was decided to test Veno further at a bidirectional packet loss rate of 7%, with the 100 Mb transfer taking 201 seconds, which is still better than the performance of Reno at a 6% packet loss rate. For completion Reno and BIC were also tested at a 7% bidirectional packet loss rate; however the transfer times for both were well over 300 seconds, at which point the connections required manual termination via the iperf tool. These observations of Veno support previous claims of enhanced performance over existing algorithms in conditions where random packet losses are prevalent. This is because a Veno sender is able to distinguish between congestive-losses and random losses, thereby reducing its *cwnd* only slightly for random losses with the aim of maintaining a higher sending rate. Since the experiments consisted only of random packet losses

being induced, Veno was able to send more data per RTT due to a larger average *cwnd*.

5.5.1 Sender Congestion Window Dynamics

To highlight the impacts of bidirectional packet losses on a TCP sender's *cwnd* behaviour, plots of the *cwnd* evolution under varying packet loss rates (0.001%, 0.01%, and 0.1%) for unidirectional versus bidirectional packet loss scenarios are presented in Figures 5.8 to 5.10 (for Reno), Figures 5.11 to 5.13 (for BIC), and Figures 5.14 to 5.16 (for Veno). These plots may help to justify the reasoning behind the argument of not assuming an error-free reverse path for TCP connections. All plots have been generated from actual values of the *cwnd* size variable (in bytes) that was read from the TCP sending machine's instrumented Linux kernel during live experiments.

Figures 5.8, 5.11, and 5.14 are plots of the behaviour of the sender's *cwnd* at a random packet loss rate of 0.001% using Reno, BIC, and Veno, respectively. The plots clearly show the differences in the evolutions of the *cwnd* under each of the packet loss scenarios. As can be seen for Reno and Veno in particular, the *cwnd* size for unidirectional errors grows to a much higher value by the end of the emulation time, experiencing no packet losses. Both plots exhibit a perfect display of the slow start phase, followed immediately by the congestion avoidance phase when the *cwnd* size reaches the *ssthresh* value. In contrast, for bidirectional conditions, the *cwnd* experiences some reductions due to the introduction of packet losses on the reverse path of ACKs, and is generally starved of growth throughout the connection. Lost ACKs will cause the sender to either timeout if the ACK does not arrive within the RTO timer value, or if three DUPACKs are received, whichever comes first. Looking at Figure 5.11 for TCP BIC, although both scenarios produced similar behaviour of the *cwnd* at a packet loss rate 0.001%, it can be seen that a *cwnd* reduction occurred earlier on for bidirectional conditions.

Increasing the random packet loss rate tenfold to 0.01% produced the plots that further supports the theory above, as presented in Figures 5.9 (Reno), 5.12 (BIC), and 5.15 (Veno), where the differences between unidirectional and bidirectional packet losses

are more pronounced. In Figure 5.9 the Reno sender experiences four *cwnd* reductions for unidirectional packet losses in comparison to six reductions for bidirectional packet losses. The Veno sender also experiences similar behaviour to Reno. Figure 5.12 for the BIC sender produced a more significant result, clearly showing the starved growth of the *cwnd* in bidirectional packet loss conditions, with a much smaller average size throughout the entire connection due to nine reductions in comparison to just five for unidirectional packet losses. As can be seen from Figure 5.12, the BIC sender's *cwnd* in bidirectional packet losses did not exceed a size of 600000 bytes; contrasting that with unidirectional packet losses, the *cwnd* size did not fall below 600000 bytes, reaching a maximum size of 1458136 bytes.

When increasing the random packet loss rate to 0.1% the *cwnd* evolution for both scenarios across all TCP senders became erratic and difficult to follow. The plots are presented in Figures 5.10 (Reno), 5.13 (BIC), and 5.16 (Veno). The first thing that stands out is the average size of the *cwnd* throughout the connections, which are all on a much smaller scale than previous plots. The Reno and Veno senders had an average *cwnd* size of just 50000 bytes, whereas BIC was able to produce a slightly higher average *cwnd* size. Focussing on the difference in the number of *cwnd* reductions as a ratio between unidirectional to bidirectional packet loss conditions, Reno experienced a 39:64 ratio, BIC experienced a 41:73 ratio, and Veno experienced a 29:61 ratio of *cwnd* reductions between the two conditions. Looking at these ratios, it is clear that on average there are twice as many *cwnd* reductions in bidirectional channel packet losses than unidirectional packet losses, which is a significant difference. This ultimately leads to lower overall TCP throughputs for the end-users, as mentioned in a previous section. Again, it highlights the importance of the reverse channel for TCP experiments (and the importance of ACKs), and how unidirectional packet loss conditions can significantly over-estimate the performance of TCP.

5.6 Chapter Conclusions

The results obtained and presented in this chapter provide strong evidence that there are significant differences in the performance of TCP between differing experimental path-loss conditions, regardless of the sender-side TCP variant used. The significant impacts of excluding a packet loss model on the feedback channel for TCP

experiments have been clearly shown through a comprehensive set of experiments over an emulation platform using real implementations of TCP. The general consensus is that in experiments where packet losses are induced only on the forward data channel, the performance results for TCP are over-estimated. In reality, over wireless channels, TCP experiences losses in both directions; hence ACKs on the reverse channel are also affected. Such bidirectional packet losses were induced into the experiments of this chapter, and the performance of TCP was shown to be worse-off. Note that although the reverse packet loss rates in this chapter may represent an upper-bound on TCP performance in bidirectional error conditions, problems do exist, especially within the 2% operating range that is typical of indoor wireless channels.

In summary then, end-to-end experiments with different 'flavours' of TCP were conducted over varying random packet loss rates, and each has been consistent in highlighting the same problems, albeit with individual characteristics. Overall, the observations are that TCP Veno proved to be the most robust in such random packet loss conditions, being able to sustain a higher packet loss rate.

Finally, the message to researchers of reliable transport protocol issues for wireless networks is that all protocol experiments, be it via simulation or emulation, should be configured to use forward and reverse channel loss models in order to fully justify the capabilities of the protocol.

Chapter 6

A Testbed for Evaluating TCP over 802.11 WLANs

6.1 Introduction

Recently there has been a great deal of research conducted into the performance issues and behaviour of TCP over wireless networks, as documented in Chapter 3, and researchers are always keen to experiment with the protocol in a variety of ways. TCP is still the most popular transport protocol in the Internet, carrying a huge proportion of traffic to end-users. The difference now is that these end-users are increasingly becoming mobile and wireless; many homes and offices are now fully equipped with WLANs [47], with the IEEE 802.11 standard being the most prominent, showing no signs of a slow-down with this trend in the foreseeable future.

TCP is a complex transport protocol that has gone through several evolutions in recent years [8], constantly being updated to meet the needs of the changing Internet and its end-users. It was originally designed as a reliable connection-oriented end-to-end transport protocol for the fixed wired Internet, at a time when its heavy usage over wireless links was not envisioned [11]. Today, the situation is somewhat very different, with the pervasiveness of wireless technologies changing the final delivery process of TCP data. It is therefore very important to gain accurate insights into the capabilities of sender-side TCP implementations over heterogeneous paths, where wireless links become part of the end-to-end journey of TCP segments.

In this chapter, the focus is on the scenario that is more common today: end-users of the IEEE 802.11 WLANs accessing the Internet and downloading content and services using legacy protocols via local APs (which act as private gateways into the wired domain). These 802.11 wireless end-users are effectively TCP clients from the perspective work in this thesis. Figure 6.1 illustrates the scenario that is becoming more commonplace today. Hence, this chapter proposes a wired-to-wireless testbed that allows for the complete experimental testing of TCP, and for the observation and evaluation of end-to-end TCP performance when segments traverse both wired and

wireless links in the same connection, where the wireless path is last-hop real-world 802.11 WLAN.

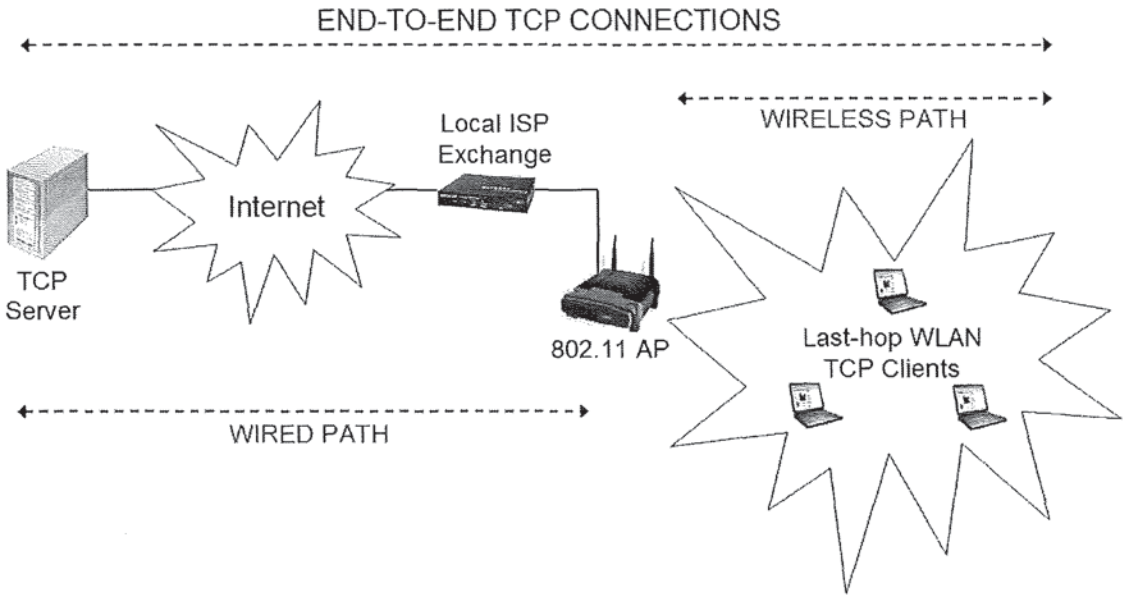


Figure 6.1: The typical scenario today for TCP senders running over last-hop WLANs

Researchers commonly use simulation as the preferred method of putting TCP under test due to the ease of setting up a wireless network environment and running experiments under controlled circumstances. NS-2 and OPNET Modeler™ both offer excellent wireless models, together with accurate implementations of TCP. However, in light of many of the advantages of network simulation, modelling the true behaviour of TCP over wireless networks is a very precise procedure, and one that simply cannot be tested using simulation alone. It has been noticed that in many research reports the conditions in which TCP is put under test have been overlooked. Assumptions are easily made regarding parameters and measurements when running simulations, and there was a lack of parallelism with how TCP is actually used and operates in the real-world today.

The idea of this work is to promote experiments on TCP over wireless networks using alternative techniques, which can only lead to a more robust TCP that will survive well into the future. This is why the proposed testbed is highly relevant and beneficial to TCP research for combined wired-to-wireless networks. The testbed can provide insights into how the wireless channel behaves, and how this affects the performance

of TCP back at the sender in the wired domain. In addition, it allows the capturing of reverse channel traffic from both the wired and wireless domains from within the same experiment, increasing the accuracy of any evaluation work on captured data. For example, using the captured data from the testbed, evaluations can be made of the sender's *cwnd* performance, accurate 802.11 frame error distributions over the WLAN can be inferred, and the relationship between TCP and 802.11 MAC can be investigated, and so forth.

6.2 Related Work and Motivations

More often than not, simulation has always been welcomed as a first choice by researchers, despite their attentiveness to the intrinsic inaccuracies associated with models of wireless protocols and channels [269]. There is now a wealth of research on the experimental studies of TCP over IEEE 802.11 WLANs, as documented in Chapter 3. Literature reviews within the area revealed that there are three distinct categories that the experimental work on TCP over wireless paths fall into; i) simulation [125] [270], ii) emulation [59] [271] [272], and iii) testbeds [41] [68] [129] [157] [273] [274] [275] [269]. It was also noticed that the network topologies used for testbed experiments are generally unrealistic of how TCP is typically used in the real-world. There appears to be the lack of an Internet component between the TCP server and the 802.11 WLAN, which was the case in [41] [129] [275].

Focussing only on testbeds for the purposes of this chapter, the work in [129] was some of the earliest contributions to the field of TCP over wireless environments. The authors in this paper used a testbed to run TCP experiments between a fixed server in the wired domain, and a client in the wireless domain. Although the work in the paper was original at the time, the testbed used was not reflective of real-world conditions today, i.e. the TCP server was positioned within the same wired subnet as the wireless AP. This is not the most accurate way of testing TCP sender behaviour, as servers today tend to be separated from the AP by the Internet. By factoring in the presence of an Internet between the server and client within the testbed, TCP segments would be subjected to additional delays or congestion-related losses that normally occur in the fixed wired domain [276]. This leads to increased levels of realism and accuracy for end-to-end TCP experiments over the testbed.

A fairly recent paper that used the idea of a testbed for evaluating TCP over wireless networks is presented in [273]. Although this is a good example of how testbeds can be very useful, the authors in this paper used the *GPRS* wireless technology for the wireless portion of the testbed, which is somewhat different to the behaviour of the 802.11 WLAN standard for supporting TCP clients.

In [274], a testbed is used for studying TCP performance over an 802.11 WLAN, but the authors' focus is primarily on ad-hoc wireless links between devices in the WLAN, with an emphasis on device mobility and roaming. There was also a lack of evaluation work on the actual behaviour of TCP at the sender-side, as the testbed lacked the ability to collect a wide range of data.

In [157] the authors demonstrated the effective use of a testbed for TCP experiments over an 802.11 WLAN. The work in this paper was clear and concise, agreeing with the motivations and views of this chapter, in that simulations do not always reveal true TCP behaviour to that observed in real-world conditions. The limitations of the work come from the fact that the authors' focus on TCP was mainly on the client-side, i.e. when wireless end-users are the TCP senders creating uplink traffic. Our view on the work of this paper is that the bulk of TCP data sending is done by servers in the Internet, and hence it is more beneficial to study TCP settings and behaviour at the server in the wired domain of testbeds, i.e. when wireless end-users are downloading data from the Internet creating downlink traffic over the WLAN.

An integrated wired-to-wireless testbed was proposed in [277], which takes into account the Internet component between the TCP senders and wireless end-users. However, the authors did not use a real-world fully operational wireless network for the wireless path; instead they used an emulated wireless channel, which accurately reproduced a *UMTS* wireless propagation environment. In addition, the testbed required the connecting of components across different sites over a *VPN* connection, with the use of a web-interface to manage and control experiments. This is a rather cumbersome approach, as it results in limited control of the testbed, and suffers from not being straightforward for other researchers to easily adopt.

Finally, other wired-to-wireless testbeds tend to be very complex arrangements, built using an array of dedicated components (including both expensive hardware and bespoke software) by researchers for a specific scenario [68] [269]. Although these testbeds tend to be very comprehensive at producing results, they often lack the 'ease-of-use' factor which is important if the testbed is to be adopted by a wider research community. It is often difficult for other researchers to gather all the necessary components in order to build such complex testbeds.

6.3 Proposed Experimental Testbed and Architecture

To advance on the aforementioned testbeds a decision was made to construct a wired-to-wireless testbed for TCP that allows certain conditions to be controlled (as in a simulator or emulator), but also one that utilises real-world protocol implementations, real-world links (both wired and wireless), real networking hardware components, and with the presence of an Internet component (via emulation) between the server and wireless clients. The idea is to minimise the level of assumptions that would need to be made when conducting experiments.

Inspiration for the testbed design came from a review of statements in [108], in which the authors discuss best practices for TCP experiments, especially on how to collect data accurately from both sides of a connection. So, unlike previous comparable testbeds, the objective here was to be able to capture data from all angles of an experiment, trying to incorporate the advantages of simulation into the testbed; specifically, to be able to capture accurate statistics from the sending TCP server in the wired domain, which is difficult to do in the real-world where the TCP server is often in a geographically distant location (i.e. in the Internet). This is a particularly unique feature of the proposed testbed, which will be elaborated upon in the coming sub-sections.

To gain a full understanding of the wireless side of experiments, a further objective was to be able to capture as much information as possible from the IEEE 802.11 WLAN whilst running end-to-end TCP experiments. Further inspiration came from the work undertaken in [73] [76] [130] [131] [240], all of which provide good insights into the techniques for capturing and analysing wireless network traffic, especially for

defining accurate loss models of the real-world. Such data could assist researchers with defining more realistic wireless models for end-to-end TCP performance testing over wireless networks.

A final design goal was that the testbed should be relatively straightforward to build using easily available hardware and software, allowing for easy repeatability of experiments, alongside offering high levels of realism and accuracy for dependable results.

6.3.1 Testbed Architecture

The proposed testbed is based on a collection of components that have each been carefully selected. The architecture of the testbed can be broken down into five separate components; i) the wired TCP server, ii) the Internet emulator, iii) the IEEE 802.11 AP (i.e. the WLAN), iv) the wireless TCP client/s, and v) the 802.11 *sniffer* devices for wireless traffic. At a high-level, there is a wired path and a wireless path for all TCP connections. Figure 6.2 illustrates the schematic layout of the testbed.

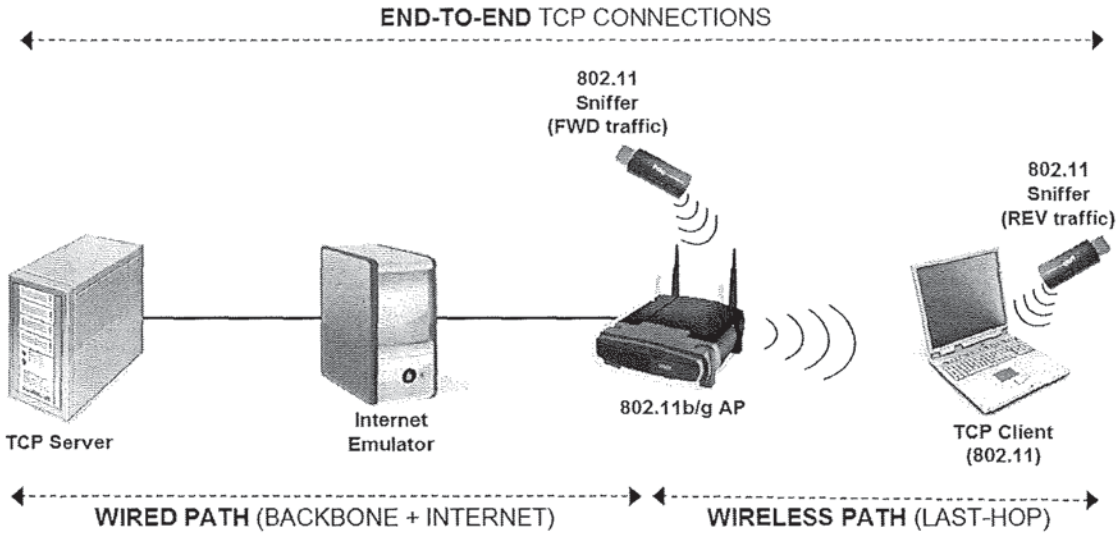


Figure 6.2: The proposed wired-to-wireless TCP experimental testbed

The TCP server and the Internet emulator both need to be fast *Pentium 4* or equivalent machines, equipped with a minimum of 1024 Mb of system memory each. The emulator machine consists of two Ethernet 10/100 NICs to allow traffic to flow

through it in both directions at layer-2 (via a bridging of the two NICs – see subsections below).

The 802.11 AP is a Linksys WRT54G device whose factory firmware was flashed with the *DD-WRT v23* [278] third-party firmware, which is still based on the IEEE 802.11b/g MAC standards, but offers many additional and advanced features for greater control in experimental situations. In a nutshell, the DD-WRT firmware was developed and released under the terms of the *GPL* for many IEEE 802.11-compatible APs based on either a *Broadcom* or *Atheros* chipset.

The TCP client/s in the WLAN need to be *Intel Centrino* or equivalent specification notebooks/machines, equipped with a minimum of 512 Mb of system memory and an IEEE 802.11b/g-compatible client adapter. However, there are no real strict requirements for the clients in the WLAN of the testbed, as in the real-world there is little control over the types of 802.11 devices that connect to the Internet, and the operating systems they run.

The 802.11 sniffer devices are inexpensive proprietary USB adaptors supplied by *CACE Technologies*, referred to as their *AirPcap* device [279]. AirPcap enables the capturing of 802.11 frames, including control frames, management frames, and low-level power information per frame. Being a completely passive capture solution, it is a great component in the testbed for capturing forward and reverse channel wireless traffic. Both AirPcap devices need to be plugged into independent MS Windows based machines for their operation.

Finally, all wired connections between the server, Internet emulator, and the AP are made using full-duplex cross-over Ethernet 10/100 cables to eliminate the need for additional hardware such as switches or hubs.

6.3.1 TCP Server Settings

The operating system of the TCP server machine needs to be running the current generation of Linux version 2.6, preferably release v2.6.13 of the kernel and newer. As already touched on in a previous chapter, from kernel v2.6.13 onwards, Linux supports

sender-side 'pluggable' congestion control algorithms for the TCP stack [81]. This allows for the switching between the different implementations at runtime from the Linux user-space by making `echo` calls to write values to the `/proc/sys/net/ipv4/tcp_congestion_control` entry. For example, the following command would set the current TCP congestion control algorithm to the Hybla variant with immediate effect:

```
shell> echo "hybla" > /proc/sys/net/ipv4/tcp_congestion_control
```

Table 6.1 summarises the TCP congestion control algorithms that are currently available for use, based on v2.6.19 release of the kernel. The implementations are being updated all the time with each new version releases of the kernel.

To extract accurate and meaningful data during experiments directly from the TCP layer, the Linux v2.6 kernel needs to be instrumented using the *web100* patches [280], which are freely available from the official web100 project website [281]. A web100-instrumented TCP stack allows low-level data from the Linux kernel to be extracted and collected at the user-space level.

The entire web100 suite is distributed as two components: i) a Linux kernel patch that implements the instruments, and ii) a set of user-level libraries and tools for reading and writing values to/from the kernel instruments, referred to as the *userland* tool. These two components must be downloaded and installed separately. While the kernel patch may be used in its entirety, the userland tool requires the web100 kernel patch to be installed for it to work. Note that the entire set of TCP instruments has been defined in an *IETF* internet-draft; they are referred to as the *kernel instrument set* (KIS) variables. The KIS variables are fully documented in the *tcp-kis.txt* document that is distributed with the kernel patch.

In summary then, the web100 instrumentation solution is a passive technique for capturing low-level information on passing traffic through the TCP-layer, as well as extracting useful data from the TCP-layer itself [280] [282].

Variant	Target Usage	Reference
Reno	Legacy Implementation of TCP	[99]
High-Speed TCP	Large Bandwidth and RTT Paths	[283]
H-TCP	Large Bandwidth and RTT Paths	[284]
Scalable TCP	Large Bandwidth and RTT Paths	[285]
BIC	Large Bandwidth and RTT Paths	[268]
CUBIC	Large Bandwidth and RTT Paths	[286]
Hybla	Transmissions over Satellite Links	[208]
Low Priority TCP	Low Priority Transmissions without Disturbing other Flows	[287]
Vegas	Estimating Path Conditions using RTT	[193]
Veno	Transmissions over Wireless Links with Random Losses	[199]
Westwood+	Large Bandwidth and RTT Paths with Random Losses	[195]

Table 6.1: Currently available TCP congestion control algorithms in Linux v2.6.19 kernel

6.3.1.1 Capturing and Analysing TCP Traffic

To generate TCP traffic flows (i.e. connections) between the server machine and the 802.11 WLAN devices in the testbed, a modified version of the popular iperf traffic measurement tool was installed on the server machine, whilst the devices in the testbed have the standard version of iperf installed. Modifications were necessary to ensure that iperf delays the sending of any data by 2500 ms after initiating a new TCP connection with the receiving end. This was to allow sufficient time for a web100 `connection_id` to be written to the kernel `/proc/web100/*id` entry, and for

the userland tool to be initiated so that it was waiting for the start of reading TCP KIS variables from the very first byte of TCP data sent. The source code of the modified iperf tool for the benefit of web100 experiments is available upon request.

A further tool called *readiperfport* was developed to work in conjunction with the userland tool after it is initiated. Userland comes ready with a set of web100 libraries that have been written in the *PythonC* language. This essentially provides a clean object-oriented interface to the kernel web100 KIS variables through userland at the user-level. The readiperfport tool, written in PythonC, takes as its initial input parameter the TCP sender's *port* of the newly created connection by iperf, which is port 5001 by default. Hence, readiperfport uses the value 5001 by default to get the corresponding web100 `connection_id`, which it then uses to read all the KIS variables from the `/proc/web100/*id` entry mentioned above. The readiperfport tool is easy to use and very customisable. It can read values from the KIS variables at regular intervals during a live TCP connection on the order millisecond granularities, which can be set by the user. The user can also specify the KIS variables they would like to read from, and where the data should be stored. By default the readiperfport tool dumps all read data into a standard text-file by the name of `web100_<connection_id>.txt` in the current directory. The source code of the readiperfport tool for the benefit of web100 experiments is available upon request.

As a final step, in order to capture all TCP data segments and ACKs that have been sent and received, the popular *tcpdump* tool [288] was installed on the server machine and on the 802.11 clients in the WLAN of the testbed. It needs to be initiated on both sides prior to the invocation of iperf, and should be configured to listen for all TCP traffic that is associated with the iperf port: port 5001 by default. The rationale behind this is to have information from both sides of a TCP connection, as recommended for accurate TCP analyses in [108]. This technique allows for the tracing of segments and ACKs from both the server's viewpoint and the 802.11 client's viewpoint. The two independent tcpdump capture files give a complete set of data for inferring accurate statistics such as the true number of segment losses encountered at the TCP-layer. The *tcptrace* tool [289] is recommended for parsing and analysing the tcpdump capture files.

6.3.2 Internet Emulator Setup and Configuration

In order to incorporate the idea of an Internet component in the testbed between the server and the 802.11 AP, a separately running emulation machine needs to be built and configured to act as a WAN traffic shaper. The idea is to be as realistic as possible in subjecting forward and reverse channel TCP flows to the typical characteristics of the Internet.

The Internet emulator machine (running a Linux v2.6 kernel release) consists of two 10/100Mbps Ethernet NICs, which must be bridged at layer-2 using the Linux bridge packages [264], as discussed in Chapter 5. This allows traffic to flow through the bridge interface in both directions (full-duplex) as close as possible to the PHY, with the objective of trying to minimise packet-header overheads. The idea behind this is to make TCP (at layer-4) believe that all sent and received segments have just traversed a wider area network, i.e. the Internet.

Ideally the emulator machine should be a fast Pentium 4 or equivalent, and have at least 1024 Mb of system memory, preferably more. To configure the machine as a WAN traffic shaper the open-source *linux advanced routing and traffic control* (LARTC) packages [290] need to be installed and configured. The key component of the LARTC package is called *traffic control* (TC), which allows a Linux machine to perform various methods for classifying, prioritising, shaping, and limiting both inbound and outbound traffic. Further components of the LARTC suite consist of netem, which was introduced in Chapter 5, and *ebtables* [291] which enables the basic Ethernet frame filtering at the bridge.

To setup the emulator machine to behave as the Internet, LARTC parameters need to be set manually by the user. Rather than inputting arbitrary values as parameters, some real-world studies were conducted on the general path characteristics of the Internet, as instructed in [8]. Primarily the interest was in those statistics that would impact the end-to-end behaviour of TCP. Hence, the popular *pchar* tool [292] was used to gauge chosen statistics from the live Internet at random intervals over a period of seven days, targeting an extensive list of popular web-servers at varying physical distances at different times of the day, including the likes of *Google.com*, *Sourceforge.net*, and

Yahoo.com. All pchar tests were performed from within a home environment using an 802.11 WLAN device connected to the Internet (via a broadband connection) as the initiating client.

The key pchar statistics that were of interest are a) the end-to-end propagation delay (including the standard deviation), b) the bottleneck bandwidth across the Internet (upstream and downstream), c) average queuing delay at intermediate hops, and d) the packet loss probability. All collected results were averaged, and the values recommended as good LARTC parameters for the Internet emulator in the testbed are a) 108.76±4 ms as the one-way propagation delay for packets (to two decimal places), b) 5.029 Mbps as the downstream bottleneck bandwidth and 0.686 Mbps as the upstream bottleneck bandwidth, c) 0.0048 ms as the average queuing delay for packets, and d) 0.002 as the probability of losing a packet. It was reassuring to know that the values obtained for the packet loss probability closely matched that of the Internet studies conducted in [276]. Once these values were applied to LARTC at the bridge interface, it enforces the rules on all forward and reverse traffic so that TCP connections were also subjected to fixed-network related issues such as path congestion, segment losses, and queuing delays as in the real Internet.

6.3.3 The Last-hop 802.11 WLAN

The wireless portion of the testbed in Figure 6.3 is a fully operational real-world 802.11 WLAN, acting as the last-hop link for TCP data segments arriving from the wired server. All TCP data segments arrive at the 802.11 AP from the wired side and are encapsulated in 802.11 data frames before being transmitted over the WLAN. TCP ACKs are generated by the TCP-layer of the 802.11 end-device/s, which are encapsulated as 802.11 data frames at the MAC before being transmitted back to the AP over the WLAN. The AP then forwards all TCP ACKs to the wired side. Figure 6.3 illustrates the flow of TCP traffic between the wired and wireless domains, defining the meanings of downlink and uplink traffic over the 802.11 WLAN.

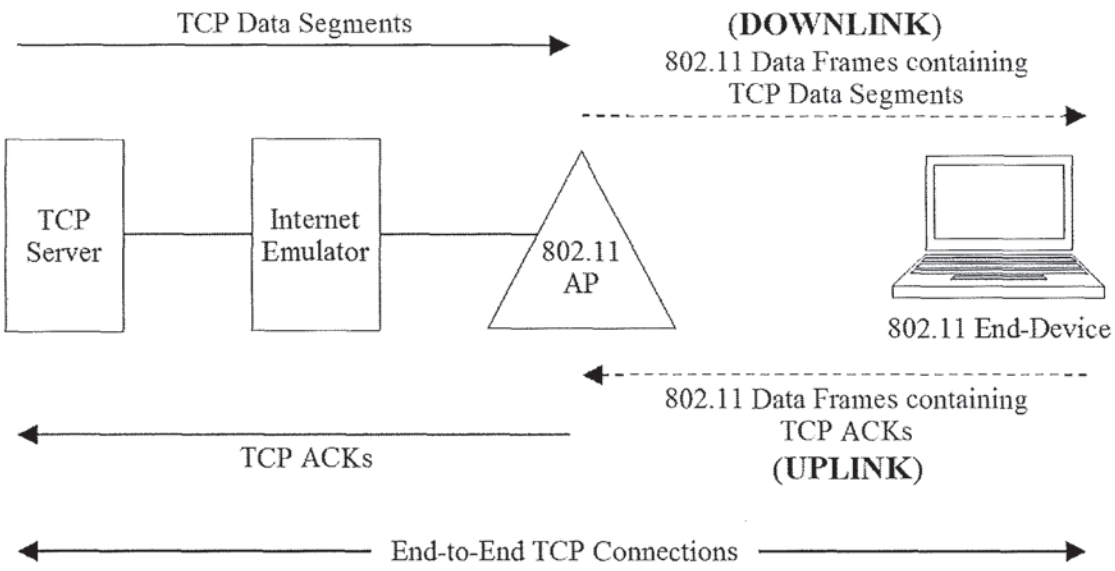


Figure 6.3: The flow of TCP traffic over the wired-to-wireless testbed

6.3.3.1 Capturing 802.11 Traffic

The AirPcap USB devices are used to capture all downlink (forward channel) and uplink (reverse channel) 802.11 traffic over the WLAN. As shown in Figure 6.2, one AirPcap device is positioned directly next to the AP so that it can capture all data frames being sent towards the end-device using downlink filters (i.e. TCP data segments), and one AirPcap device is positioned directly next to the end-device so that it can capture all data frames being sent back to the AP using uplink filters (i.e. TCP ACKs). This technique produces two independent AirPcap capture files for each TCP experimental run, a downlink traffic dump and an uplink traffic dump. The capture dumps can then be analysed to infer useful statistics about the transmission characteristics of 802.11 frames over the WLAN, and about the impacts on TCP segments and ACKs that they encapsulate.

6.3.3.2 Analysing 802.11 Traffic

The analysis of each AirPcap capture dump is performed in two stages: i) the filtering stage, and ii) the processing stage.

In the filtering stage the popular *wireshark* network analyser tool [293] is used to load and display the contents of each capture file. Wireshark filtering rules are then applied

in order to display only those frames that are of interest. For example, if loading up the downlink traffic dump then only those frames with the *source address* field equal to the AP's MAC address and the *destination address* field equal to the end-device's MAC address should be displayed. Once the frames of interest have been filtered, all 802.11 and TCP protocol header field values inside each of the frames are exported to a separate text file, in the order that they were captured. Hence there will be two text-files, one for downlink traffic and one for uplink traffic. The text files are then ready to be processed and analysed.

In the processing stage the text files are vigorously analysed using a bespoke script that has been developed using the *Perl* scripting language. Hereinafter, the script will be referred to as the *analysewiresharkcaptures* tool, and is openly available to the research community upon request. The analysewiresharkcaptures tool takes as its input parameters the filenames of the downlink and uplink text-files, and after processing them it outputs the *retransmission distribution* of all frame transmissions by the 802.11 MAC, the frame error rate (FER) for 802.11 data frames for both the downlink and uplink captures, and the amount of time TCP segments and ACKs spend waiting in the *transmit queue* of the MAC before being released. The complete methodology behind how the analysewiresharkcaptures tool has been programmed to compute these statistics is presented in the subsections below.

Computing the 802.11 Frame Retransmission Distributions for TCP

Here an effective methodology for analysing the behaviour of TCP traffic over an IEEE 802.11 WLAN is presented. The methodology on captured frames produces the distribution of retransmissions for all frame transmissions by the 802.11 MAC for both the downlink and uplink channels (i.e. by the AP and by the end-device), which may be insightful to those working in the field of error control techniques.

To define, a frame error occurs when the frame is not received successfully by the 802.11 MAC layer at the intended recipient/device. A major cause of such errors are due to bit corruptions of frames whilst in transit due to noise over the radio medium; here frames will fail their 802.11 frame check sequence (FCS) check at the receiving 802.11 MAC and are immediately discarded. A further possible cause of frame losses

is when frames simply do not arrive at the intended recipient/device due to severe signal fading.

It is always the sending MAC that has responsibility for the successful transmission of data frames to the recipient. Upon the successful arrival of a data frame at the recipient MAC without errors, it immediately confirms to the sending MAC a successful delivery by transmitting an 802.11 ACK frame. Upon receiving an 802.11 ACK for the most recently transmitted data frame, the sending MAC transmits the next data frame in the sequence from its transmit queue. If the sending MAC does not receive an 802.11 ACK frame within a set period of time, it will immediately retransmit the same data frame. This is known as an *802.11 ACKTimeout*. This process continues until the number of retransmission attempts equals the retry limit of the sending MAC. At which point the sending MAC will discard the data frame in concern, and proceed to the next data frame in sequence from its transmit queue [20]. Note that a sending MAC can either be a downlink sender (i.e. an AP transmitting TCP data segments), or an uplink sender (i.e. an end-device transmitting TCP ACKs).

Based on the above description, it can be deduced that a sending MAC will be forced to retransmit an 802.11 data frame if one of two situations (or conditions) occur:

1. A data frame is lost in transmission or is received in error at the receiving MAC, in which case an 802.11 ACK will be not be generated by the receiving MAC.

2. A data frame is received successfully by recipient MAC, and an 802.11 ACK is transmitted back towards the sending MAC to positively acknowledge the successful reception. However, the 802.11 ACK frame is lost in transmission or is received in error at the sending MAC, in which case the sending MAC is still waiting for the 802.11 ACK. Eventually the sending MAC's retransmission timer will expire, causing an 802.11 ACKTimeout.

The analysewiresharkcaptures tool uses the 802.11 header sequence number field to keep track of the evolution of all frame transmissions that occurred at the sending MAC, for both downlink and uplink channels. Using the evolution of sequence numbers for data frames from both sides, the tool computes and produces the

retransmission distributions, $R_{DOWNLINK}(i)$ and $R_{UPLINK}(i)$, (for $i=1,2,...,M$), for all transmitted 802.11 data frames for both sending MACs. The value of $R_{DOWNLINK}(i)$ represents the number of data segments requiring i retransmissions by the AP before it was accepted by the receiving 802.11 MAC, and where M is the retry limit. Likewise, the value of $R_{UPLINK}(i)$ represents the number of data frames that have been retransmitted i times by the 802.11 end-device. For example, the value of $R_{DOWNLINK}(3)$ implies the total number of data frames during the entire experiment that had to be retransmitted three times by the AP before it was successfully accepted by the 802.11 MAC-layer of the end-device. Note that the value of M could be different for the AP and the end-device/s, depending on the specific implementation of the IEEE 802.11 MAC which can vary from vendor to vendor. Here it is assumed that M at both sides is equal.

The retransmission distributions, $R_{DOWNLINK}(i)$ and $R_{UPLINK}(i)$, can be used to compute the probability distribution functions for the frames in which an error occurs, $P_{DOWNLINK}(i)$ and $P_{UPLINK}(i)$, associated with the sending MAC retransmitting a data frame i times before a successful delivery, where i ranges from 1 up to its retry limit, M. Hence, $P_{DOWNLINK}(5)$, for example implies the probability of a data frame in which an error occurs being retransmitted five times by the AP, with success on the fifth attempt (i.e. six transmissions of the same sequence number in total). The value of $P_{DOWNLINK}(i)$ is computed as follows:

$$P_{DOWNLINK}(i) = \frac{i \cdot R_{DOWNLINK}(i)}{N_{APERRORS}} \qquad \textbf{(Eq. 6.1)}$$

where $N_{APERRORS}$ is the total number of data frames (i.e. TCP data segments) where an error occurred in transmission by the AP during the experiment. $N_{APERRORS}$ is calculated as shown by Eq. 6.3.

The value of $P_{UPLINK}(i)$ is computed as follows:

$$P_{UPLINK}(i) = \frac{i \cdot R_{UPLINK}(i)}{N_{CLIENTERRORS}} \qquad \textbf{(Eq. 6.2)}$$

where $N_{CLIENTERRORS}$ is the total number of data frames (i.e. TCP ACKs) where an error occurred in transmission by the 802.11 end-device during the experiment.

By having an awareness of retransmission probabilities for 802.11 WLAN channels it could help researchers working in the field of error recovery to more accurately design ARQ mechanisms that can dynamically adapt their persistence levels according to historic probabilities maintained by the MAC. They may even assist researchers of TCP; for example, if the probability values of $P_{DOWNLINK}(M)$ and $P_{DOWNLINK}(M\text{-}1)$ are quite high in relation to lower values of i (for $i=1,2,...,M$, where M is the retry limit of the MAC) then it could be inferred that the AP's ARQ mechanism at the MAC is likely to increase the overall end-to-end delay experienced by TCP data segments arriving at the AP from the fixed domain (i.e. TCP's RTT is likely to experience periods of sudden increases). By knowing this, a TCP sender's RTO timer could be heuristically modified to cater for such fluctuations in the RTT calculation, thereby avoiding unnecessary timeouts and reductions of the *cwnd* size. The same is also true if $P_{UPLINK}(M)$ and $P_{UPLINK}(M\text{-}1)$ are of high values, which would cause delays to returning TCP ACKs from the WLAN.

Calculating Downlink/Uplink 802.11 Frame Error Rates (FERs) for TCP

This subsection presents a new technique for independently calculating the total frame error rate (FER) for downlink (forward channel TCP traffic) 802.11 data frames (containing TCP data segments), $FER_{DOWNLINK}$, and the FER for uplink (reverse channel TCP traffic) 802.11 data frames (containing TCP ACKs), FER_{UPLINK}.

To compute $FER_{DOWNLINK}$, the total number of downlink data frame errors, $N_{APERRORS}$, must be calculated first, and is given by the following expression:

$$N_{APERRORS} = \sum_{i=1}^{M} i \cdot R_{DOWNLINK}(i) \qquad \textbf{(Eq. 6.3)}$$

The value of $FER_{DOWNLINK}$ (%) is then calculated using the following expression:

$$FER_{DOWNLINK} = \frac{N_{APERRORS}}{N_{APTOTAL}} \cdot 100 \qquad \textbf{(Eq. 6.4)}$$

where $N_{APTOTAL}$ is the total number of data frames that are transmitted by the AP during an experiment, including all retransmissions, as shown by Eq. 6.5.

$$N_{APTOTAL} = \sum_{i=0}^{M} (i+1) \cdot R_{DOWNLINK}(i) \quad \textbf{(Eq. 6.5)}$$

To compute FER_{UPLINK}, the total number of uplink data frame errors, $N_{CLIENTERRORS}$, must be calculated first, and is given by the following expression:

$$N_{CLIENTERRORS} = \sum_{i=1}^{M} i \cdot R_{UPLINK}(i) \quad \textbf{(Eq. 6.6)}$$

The value of FER_{UPLINK} (%) is then calculated using the following expression:

$$FER_{UPLINK} = \frac{N_{CLIENTERRORS}}{N_{CLIENTTOTAL}} \cdot 100 \quad \textbf{(Eq. 6.7)}$$

where $N_{CLIENTTOTAL}$ is the total number of data frames that are transmitted by the 802.11 end-device during an experiment, including all retransmissions, as shown by Eq. 6.8.

$$N_{CLIENTTOTAL} = \sum_{i=0}^{M} (i+1) \cdot R_{UPLINK}(i) \quad \textbf{(Eq. 6.8)}$$

Transmission Delays at the 802.11 MAC for TCP Traffic

One major problem for TCP includes spurious expirations of its RTO timer due to the variable (and often lengthy) delays over the wireless path of a connection, causing unnecessary TCP retransmissions and reductions of the sender's *cwnd* size. Delays over the WLAN are variable because the 802.11 MAC protocol uses its own stop-and-wait ARQ technique to retransmit locally unacknowledged frames before continuing with the next frame in the sequence [23]. This can suddenly increase TCP's RTT calculation, as well as lead to a RTO event. Such problems exist because TCP is disconnected from all error recovery mechanisms performed below layer-4, and has no control over the delays experienced by individual segments or ACKs.

This subsection therefore presents the methodology used to calculate the delays experienced by TCP data segments and ACKs being transmitted by an 802.11 MAC until each frame is successfully delivered, accounting for frame retransmissions.

Assuming that one TCP data segment is encapsulated by just one 802.11 data frame, and that one TCP ACK is also encapsulated by just one 802.11 data frame (i.e. there is no *fragmentation* at the MAC), then it is possible to compute the individual delays experienced by each unique TCP data segment and ACK before it is successfully received by the MAC of the recipient device.

The analysewiresharkcaptures tool parses the downlink and uplink capture text-files to extract values from the *timestamp* field in the 802.11 header of each data frame, which is set at the point when it is actually transmitted by the sending MAC. The tool also extracts the corresponding 802.11 sequence number of the outgoing frame. Both values are immediately added to separate arrays, one for the timestamp (`timestamp[]`), and one for the sequence number (`sequenceno[]`).

After parsing each file from beginning to end, each array is then processed. The first element of `timestamp[]` is stored in a temporary variable, `starttime`, and the first element of `sequenceno[]` is stored in a temporary variable, `currentseqnum`. The tool then reads the second element of `sequenceno[]`, and stores its value in `nextseqnum`. If the value of `nextseqnum` and `currentseqnum` are equal then the tool moves onto the third element of `sequenceno[]`, storing its value `nextseqnum`. If the value of `nextseqnum` and `currentseqnum` are still equal then the tool moves onto the fourth element of `sequenceno[]`, and this continues until `nextseqnum` does not equal `currentseqnum`. At this point the corresponding element of `timestamp[]` is read and stored in the variable `endtime`.

The difference between `endtime` and `starttime` is then calculated, and the result is stored in another array, `delays[]`. This value is effectively the period of time taken to successfully transmit the 802.11 data frame with sequence number

`currentseqnum`; hence it can also be said that it is the delay experienced by the TCP data segment or ACK over the WLAN.

Finally the value of `currentseqnum` is set to `nextseqnum`, and the tool continues traversing through `sequenceno[]` after setting the `starttime` variable to the next in-sequence `timestamp[]` element. Once all processing is complete, the tool outputs all elements of the `delays[]` array for each of the text-files into two separate output text-files, one for downlink TCP data segment traffic and one for the uplink TCP ACKs. Each text-file contains a list of all the individual time periods that will have been experienced by TCP traffic over the WLAN.

Such insights can be useful to researchers wanting to gain insights into the pattern of delays experienced by downlink and uplink TCP traffic over 802.11 WLANs under specific conditions. The data could assist with the better tuning of the sender's RTO timer in-line with the likelihood cause of an unexpected hike in the TCP RTT. A likely sender could use the delay distributions (of either the data segments or the ACKs) to make decisions as to whether the delay could be due to network congestion in the fixed path, or whether it is due to delays at the 802.11 MAC as a result of poor channel conditions and constant frame retransmissions. The key idea is to prevent unnecessary timeouts and retransmissions at the sender when the 802.11 MAC may have already successfully delivered the TCP data segment in concern.

6.4 Evaluating the Impacts of an 802.11g WLAN on TCP

In this section the results of extensive real-world measurements made over the proposed wired-to-wireless testbed are presented, which consisted of a TCP sender combined with a last-hop real-world indoor IEEE 802.11g WLAN. This section also acts as a demonstration of the usage of the testbed.

The experiments investigate the effects of radio signal attenuation due to varying distances of an 802.11 end-user from the AP. The focus is on the downlink and uplink FERs for TCP flows over the WLAN, and consequently on the retransmission

behaviour of the fixed TCP. This is one the strengths of the testbed, as it is able to gauge TCP and 802.11 statistics from the same experiment.

Specifically, a series of experiments are performed using the different modulation schemes belonging to the 802.11g OFDM PHY to gauge potential differences in characteristics between them. For each modulation scheme, a separate downlink and uplink FER against varying SNR is calculated from actual captures, clearly distinguishing between the two.

Finally, insights into the real-world probability distributions of the number of retransmissions per data frame that were made over the WLAN by the AP are also given, with useful findings.

6.4.1 Motivations

It is important to segregate the forward and reverse channels of the 802.11 WLAN for TCP studies because each direction is carrying a different type of TCP segment, leading to differing reactions by the TCP sender when losses or delays occur. When 802.11 data frame losses occur in the forward channel of the WLAN they affect TCP data segments, which then lead to the TCP-layer of end-device generating DUPACKs for each subsequently arriving TCP data segment. When three DUPACKs reach the sender, the fast retransmit and fast recovery algorithms are invoked, causing the sender to retransmit the presumed lost data segment. Similarly, 802.11 data frame losses in the reverse channel of the WLAN affect TCP ACKs on their way back to the sender. If ACKs are lost then the TCP sender will not receive the acknowledgment that it is waiting for in order to advance its *cwnd* size and the sending of more data. However, it will only wait until the expiry of the RTO timer for the ACK to arrive, after which it simply retransmits the data segment in concern, reducing its *cwnd* size drastically. Of course, the ACK may actually arrive after the TCP sender retransmits, as it may simply have been delayed due to the end-device's MAC performing its ARQ on the 802.11 data frame containing the TCP ACK in concern.

Another problem with TCP senders is where segments (both data and ACKs) are randomly (or stochastically) lost over the wireless path, which ultimately leads to

reductions of the sender's *cwnd* size as with congestion-related losses. As discussed previously in Chapter 3, TCP losses can occur in both the forward and reverse channels of a WLAN due to a range possible causes, ranging from poor radio channels to unfairness issues between downlink and uplink flows, and so on.

Several attempts have been made to correlate the error characteristics of the 802.11 WLAN with the behaviour of TCP [36] [37] [40], however, very little literature was found that investigates the real-world error characteristics of a typical indoor 802.11 WLAN, cross-comparing it to the segment retransmission behaviour of a TCP sender, which also studies both the forward and reverse channels of the WLAN. Since the reverse channel of a WLAN also consists of 802.11 data frames that encapsulate TCP ACKs, which the sender is solely reliant upon for advancing its *cwnd* size, then it is of prime importance that the frame error characteristics of a WLAN in both channel directions was understood. Therefore, simultaneously capturing and studying data from the 802.11 WLAN and from TCP at the sender for the same experiment is what is promoted in this chapter, often referred by the community to as a *cross-layered* approach.

6.4.2 Testbed Configuration Settings

The testbed topology shown in Figure 6.2 was used for all experiments in this section, with the wireless path consisting of a real-world indoor WLAN using the 802.11g standard, using just a single stationary 802.11g end-device.

The TCP server was set to use the Reno implementation from the array of possible variants, as it reflects a widely used TCP. The SACK option, timestamps, delayed-ACKs, and window scaling extensions were also enabled, as this is typical of sender-side implementations today. The TCP MSS was set to 1448 bytes.

The 802.11g device was an *Intel Centrino* notebook with 512 Mb of system memory, and was running the MS Windows XP *SP2* operating system, which is typical of most WLAN end-user devices today. No changes were made to the TCP/IP stack on the notebook. The built-in 802.11g adaptor was an *Intel Pro Wireless 2200BG* card, with settings left to their factory defaults, which included a data frame retry limit of 15.

The IEEE 802.11 AP, running the DD-WRT v23 firmware, was configured to operate in 802.11g mode only. The firmware was also used to fix the data transmission rate (and hence the OFDM modulation scheme) of the AP to the desired value, as given in Table 6.2. All other AP settings were left to their defaults, which included an 802.11 MAC frame retry limit of 7, as specified by the IEEE 802.11g standard.

OFDM Modulation Scheme	**Supported Data Rate (Mbps)**
BPSK	6, 9
QPSK	12, 18
16-QAM	24, 36
64-QAM	48, 54

Table 6.2: 802.11g modulation schemes and supported WLAN data rates

6.4.3 Measurements and Scenario

To keep the analysis work focused, only a single 802.11g device was used in the last-hop WLAN in order to exclude the effects of multiple clients, such as 802.11 MAC contention issues and frame collisions that do occur (refer to Chapter 3). This was to capture the best-case scenario results. The aim was to investigate the error characteristics over the WLAN, i.e. to gain real-world insights into the error rates of frames traversing the forward channel and reverse channel independently. Hence the focus could be put solely on the relationship between indoor channel conditions and TCP sender behaviour in the wired domain.

The scenarios concerned were that of the effects on a TCP sender of increasing the distance of the 802.11g device from the AP. Simultaneously for each unique distance from the AP, an additional objective was to discover how each 802.11g PHY modulation scheme impacted the performance results. In order to keep the analysis work systematic, each unique modulation scheme of the OFDM system was chosen to experiment with. The chosen schemes were BPSK at 9 Mbps, QPSK at 18 Mbps, 16-

QAM at 36 Mbps, and 64-QAM at 54 Mbps, covering all of the OFDM modulation schemes.

The 802.11g device was positioned at exactly 2m, 4m, 6m, 8m, and 10m in a straight-line distance from the AP. At each of these distances there were an array of common office obstacles, including plasterboard walls, furniture, and sources of electrical and radio interference, creating LOS and NLOS situations for each case. Technically, these distances are only useful as labels for positioning purposes due to the uncontrolled nature of the WLAN environment. Refer to Table 6.3 for more details on what each of the discrete distances actually implies. Prior to commencing any experimental work, a wireless site survey was conducted from within the home-office WLAN using the popular *netstumbler* wireless network scanning tool [294] to assess the levels of interference from neighbouring WLAN APs operating in the same channel, which is a common situation today due to the popularity and penetration of the 802.11 technology. Then, using the built-in IPW2200BG card of 802.11g device (in conjunction with proprietary Intel software) a signal quality measurement survey was performed at each of the locations over a period of seven days at different times of the day, capturing metrics such as signal and noise values as reported by the device.

For the experiments, the testbed AP was set to operate in channel 11 of the 2.4 GHz band. The netstumbler tool revealed that there were 10 additional APs detected nearby, consisting of two operating in channel 1, three operating in channel 6, and five operating in the same channel 11. In Table 6.3 the averages of the signal quality measurements at varying distances from the AP are given, as perceived by the 802.11g device. Figure 6.4 is a screenshot from netstumbler revealing results of the site survey.

Distance from AP (m)	**Path**	**Signal (dBm)**	**Noise (dBm)**	**SNR (dB)**
2	LOS	-43	-93	50
4	LOS	-50	-94	44
6	LOS	-51	-93	42
8	NLOS	-56	-94	38
10	NLOS	-61	-93	32

Table 6.3: Indoor home-office WLAN signal quality measurements with varying distance

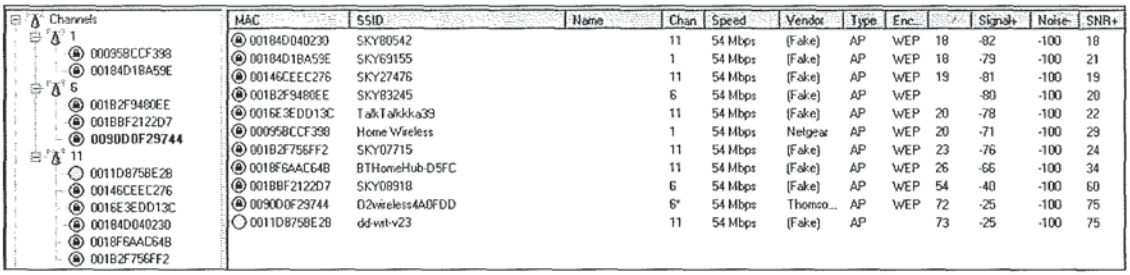


Figure 6.4: Site survey results using netstumbler WLAN scanner

For the experiments then, the scenario that was of particular interest was the performance of a wired TCP sender (i.e. on a server in the Internet somewhere) when an IEEE 802.11g end-user in a last-hop WLAN (i.e. in a typical home or office environment) downloads a 10 Mb file from the TCP sender. The primary interest was to discover whether a relationship existed between a TCP sender in a geographically different location, and the perceived channel SNR of an end-device in the last-hop 802.11g WLAN, and to investigate if the PHY modulation scheme has any impacts on the performance. Note that iperf was used to generate the TCP traffic at the server in the wired-to-wireless testbed (refer to earlier sections for more details). In total each 10 Mb transfer was run three times at each of the five locations, and repeated for each of the four modulation schemes. All results were measured in a stable and consistent indoor environment across all experimental runs. All results in the following subsection have been averaged.

6.4.4 Results and Discussions

Downlink and Uplink Frame Error Rates

Figures 6.5 and 6.6 are the plots of the forward (downlink) and reverse (uplink) channel FERs for the BPSK, QPSK, 16-QAM, and 64-QAM modulation schemes that were tested over the 802.11g WLAN as the measured SNR of the end-device was increased (i.e. decreasing distance from the AP). The results presented in Figures 6.5 and 6.6 were produced using Eq. 6.4 and Eq. 6.6 (via the *analysewiresharkcaptures* tool) based on 802.11 frame captures from the testbed. On each figure an average trend-line has also been drawn, representing a general trend of the FER across all modulation schemes.

It is evident from Figure 6.6 that the FER over the reverse channel in all experiments is very much present, and potentially an issue. Because the reverse channel consisted of 802.11 data frames encapsulating TCP ACKs, it can be said that both protocols (TCP and the 802.11 MAC) would be affected by this. The plots also show that the error rates in both channel directions are different from each other, something which should be considered when developing channel loss models for indoor WLANs. Finally, supporting the work undertaken in Chapter 5 of the thesis, a reverse channel error model should not be omitted by researchers of wireless networks.

Looking at both Figures 6.5 and 6.6, the initial observations are that frame errors are prevalent in both channel directions, even at very short distances of the end-device from the AP. Another observation is that the reverse channel possesses FERs that mustn't be ignored; the results indicate that TCP ACKs do become erroneous on their way to the AP (to be forwarded to the TCP sender), and hence can cause unnecessary retransmissions of a TCP data segment if the end-device's MAC is unable to recover the TCP ACK within a suitable time period.

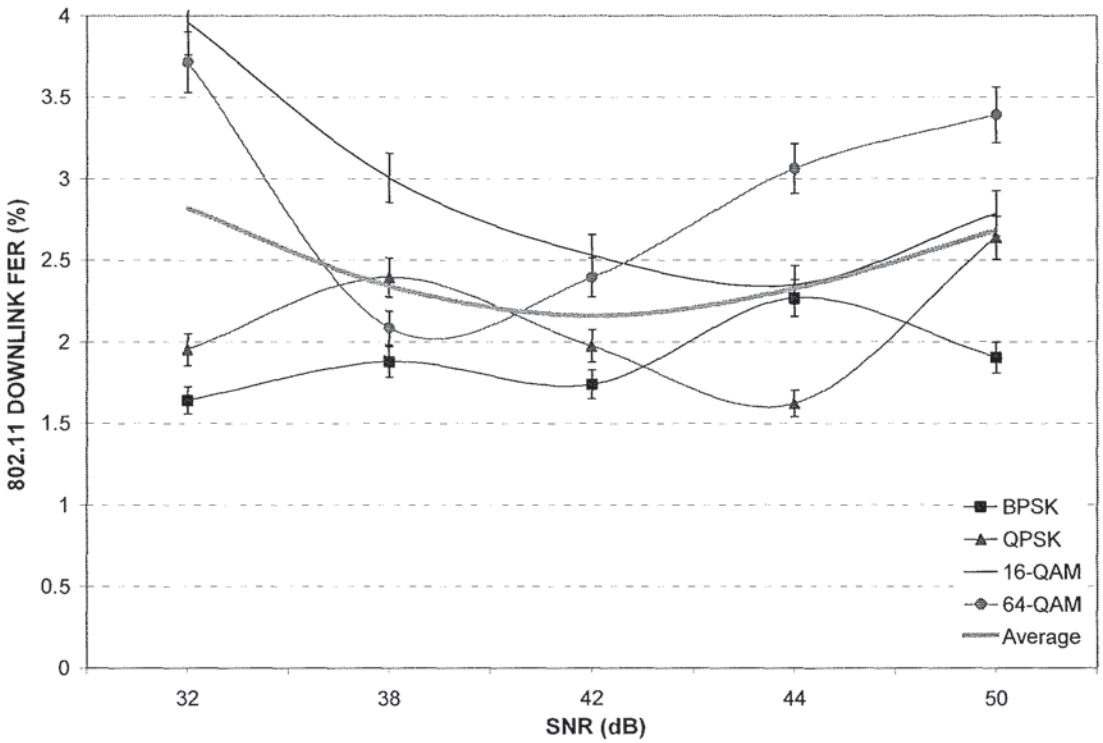


Figure 6.5: Forward channel FERs over the WLAN under OFDM modulation schemes

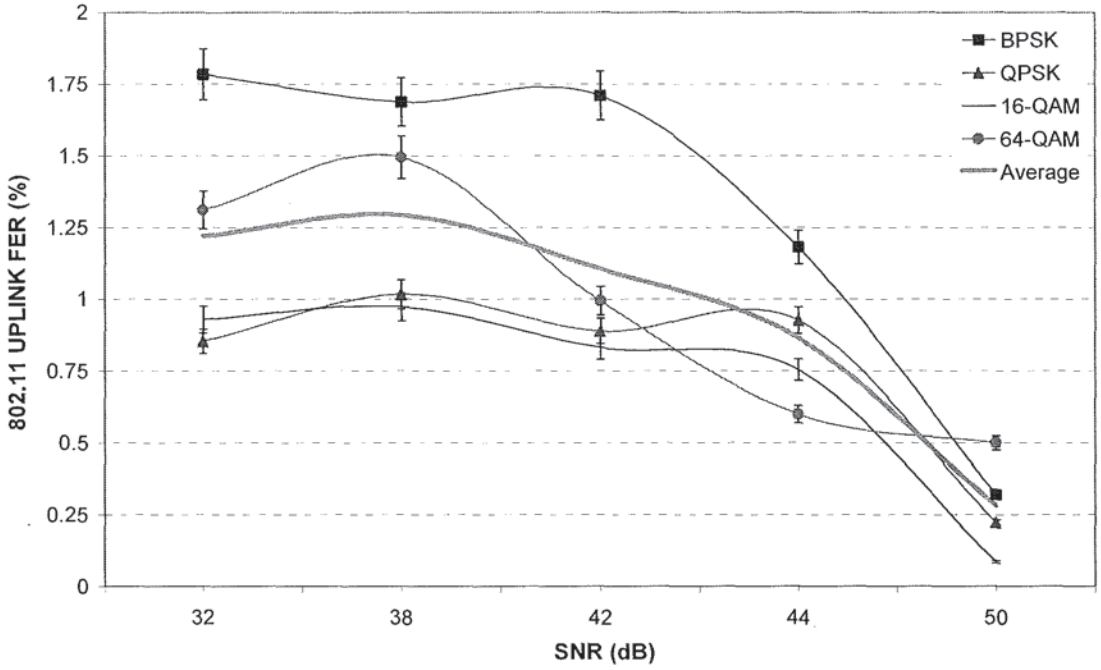


Figure 6.6: Reverse channel FERs over the WLAN under OFDM modulation schemes

Looking specifically at the downlink FERs in Figure 6.5, it can be seen that at low SNR values, with the end-device positioned no more than 10 metres from the AP, the FER can reach as high as 4% of all data frames transmitted by the AP. It was also noticed that even at very high SNR values, with the end-device positioned less than 5m from the AP the FER remained in the region of 2.5%. Overall, BPSK and QPSK performed better than 16-QAM and 64-QAM for all SNR values. However, looking at the average trend-line in Figure 6.5, the average FER for data frames in the forward channel remained fairly constant across all distances, in the region 2% to 3%.

Looking specifically at the reverse channel FERs in Figure 6.6, it can be seen that data frame errors in the uplink direction possess their own unique error characteristics. First of all, at low SNR values the FER can reach as high as 2% of all data frames transmitted, which is quite significant and comparable to forward channel FERs. An interesting behaviour that was noticed is that uplink data frames were less affected at the higher SNR values, with FERs dropping significantly in the region of 0.25% to 0.5% on average across all the modulation schemes. Surprisingly, BPSK produced the highest number of reverse channel errors, contradicting its performance on the forward channel. Notice also that 16-QAM produced the lowest number of reverse channel frame errors, but produced the highest number on the forward channel.

In summary then, the significance of 802.11 (data) frame errors has been clearly highlighted by the experimental results. It is emphasised that these results represent a best-case scenario, when in fact FERs could be much higher as soon as more 802.11 devices are factored into the WLAN increasing channel contentions, or when the distances from the AP are greater than 10 metres which is often the case. Researchers should therefore at least consider these results when designing applications for higher-layer protocols (i.e. TCP) for usage over wired-to-wireless systems.

TCP Sender Retransmission Behaviour

With accurate insights into the error characteristics of the testbed WLAN, this subsection presents the TCP sender's retransmission behaviour (extracted from the web100 kernel) for the same experiments conducted in the preceding subsection, again highlighting one of strengths of the proposed testbed. In other words, the testbed allows for the observation of how TCP's error recovery mechanism behaved in the presence of forward and reverse channel FERs in the WLAN as the distance of the end-device was altered from the AP. Thus, Figure 6.7 presents the results of the average number of TCP data segments that were retransmitted by the server during an entire 10 Mb transfer with increasing SNR over the last-hop WLAN.

Looking at Figure 6.7, the initial observation is that the TCP sender had to retransmit data segments at all SNR values for each of the modulation schemes, with 64-QAM consistently causing the greatest number of segment retransmissions. In fact, with the client located just 10 metres from the AP, 64-QAM caused on average 181 retransmissions of data segments, i.e. amounting to over 5% of the original data (10 Mb) being retransmitted. Note also that each of the retransmissions would have stalled the evolution of sender's *cwnd* size, affecting its sending rate each time.

A second observation from Figure 6.7 is that of a negative correlation for TCP retransmissions with increasing SNR. In order to get a better understanding of this behaviour these observations are cross-referenced with the forward channel (Figure 6.5) and reverse channel (Figure 6.6) FERs. It is noticed that the average reverse channel FERs also possessed a negative correlation with increasing SNR values, whereas the forward channel FERs remain fairly constant. This leads to the hypothesis

that TCP retransmissions may be more closely related to reverse channel losses, rather than forward channel losses. A possible explanation for this is if TCP ACKs are lost in the reverse channel of the WLAN, then the TCP sender in the wired domain will not receive that ACK, and eventually its RTO timer will expire, causing it to retransmit the unacknowledged data. However when cross-comparing this result with Figure 6.5 for the downlink FERs, the result of obviously contradictory to expectations, and therefore a full conclusion cannot be drawn.

A second possible explanation for linking the reverse channel FERs to TCP segment retransmissions is due to the *ACK-compression* effect occurring [11], leading to TCP data segments being dropped from the AP's sending buffer due to bursts of data being injected into the network as a result of sudden increases in the sender's *cwnd* size. TCP's self-clocking nature means that it relies on the timely arrival of ACK segments in order to progressively advance the *cwnd* size and the sending of new data into the network by probing network capacity gently. However, due to the existence of frame errors on the reverse channel of the WLAN, TCP ACKs can go missing. If several TCP ACKs are lost between two successively arriving ACKs, then when they eventually reach the sender in the wired domain, it will cause TCP to suddenly send out back-to-back all the data segments being acknowledged by the very last ACK, resulting in the ACK-compression effect. Because this phenomenon can occur in a single RTT, it could burden the entire forward path with an instant burst of new data segments. Such an event could further exacerbate conditions, especially on the forward channel, because the data segments would traverse the wired domain and arrive at the AP's sending buffer. If the buffer has already reached capacity, segments would of course be dropped, leading to further losses.

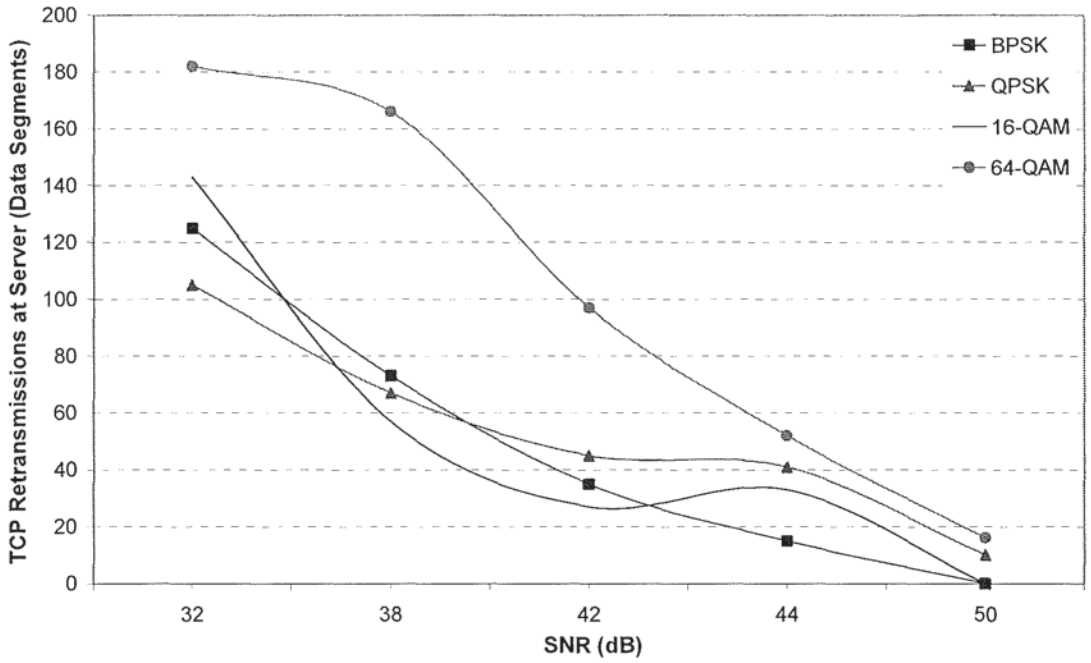


Figure 6.7: Number of data segment retransmissions by TCP at the wired server

To summarise then, the results obtained highlight the fact that TCP sender performance is not only affected by TCP data segments being lost in the forward wireless channel, but can also be attributable to TCP ACK segments being lost in the reverse channel. The significance of highlighting to researchers the prevalence of reverse channel errors is that in many cases link asymmetry is not considered in TCP loss models due to the relatively smaller size of TCP ACKs in comparison to data segments. Because TCP is an *ACK-clocked* protocol relying on the reverse channel for feedback, the sender can easily misinterpret an erroneous reverse channel as network congestion on the forward channel, thereby reducing its sending rate unnecessarily, ultimately leading to starved throughputs for wireless end-users.

Probability Distribution of Frame Retransmission Attempts by the AP

The retransmission probability, $P_{DOWNLINK}(i)$, is the probability of a frame being retransmitted i times by the AP before success, as defined by Eq. 6.1. Recall from an earlier section that i ranges from 1 to M, where M is the retry limit of the sending MAC and is equal to 7 for the AP. Hence, $P_{DOWNLINK}(4)$, for example implies the probability of a data frame being retransmitted four times, with success on the fourth attempt (i.e. five transmissions of the same frame in total).

The results presented in this subsection are from the same TCP experiments conducted in the preceding subsection, and have been averaged. All probabilities are based on the total number of data frames transmitted in the downlink and uplink channels of the WLAN per experiment. Figures 6.8 to 6.12 present the distributions of the downlink frame retransmission attempt probabilities, $P_{DOWNLINK(i)}$, for each of the modulation schemes tested at distances of the end-device of 2, 4, 6, 8 and 10 metres from the AP, respectively. The plots do not show the case where i is equal to zero, as technically these are not classed as retransmissions. All probabilities have been calculated using using $P_{DOWNLINK(i)} \times FER_{DOWNLINK}$.

Looking initially at Figure 6.8, single retransmission attempts by the AP dominated the probability distribution, with double attempts also being present. The probabilities for i ranging from 3 to 6 were very small, as would be expected in good channel conditions, with only the QPSK modulation scheme producing incidences of 5 and 6 retransmissions per frame. There were not any incidences of 7 retransmissions per frame. Figure 6.8 also shows that the 64-QAM modulation had the highest probability of single, double, and triple frame retransmissions occurring.

From Figures 6.9 (4 metres) and 6.10 (6 metres), it can be seen that there was an increased likelihood of a greater number of retransmission attempts per frame, especially for QPSK and 64-QAM, which in Figure 6.9 had a higher probability of retransmitting the same frame 6 times than when i is equal to 2.

The most striking observation can be seen in Figure 6.12, where the initial observation was that the $P_{DOWNLINK(i)}$ distribution possessed a very different behaviour to that presented in previous plots. The striking result here was that 60% of all frames sent by the AP using the QPSK modulation scheme were retransmitted 7 times on average. Note also that the AP's retry limit was set to 7, so it was possible that a subset of these frames were discarded by the AP (i.e. complete losses). QPSK also had the least percentage of retransmission lengths ranging from 3 to 6 inclusive. Equally striking was that for 64-QAM, over 50% of all sent frames were retransmitted 6 times. Of course, such events can be detrimental to TCP's end-to-end performance, as they would have increased the delay experienced by TCP data segments over the WLAN due to the persistence of the ARQ mechanism. Such delays can hike TCP's estimate of

the RTT, and cause unnecessary retransmissions due to the expiry of the RTO timer. Taking the situation where *i* is equal to 6, the AP would have succeeded in transmitting the frames on the 6th attempt, however due to the increased delays TCP may have retransmitted the same data segment from the fixed domain (and hence reduced its *cwnd* size), ultimately leading to a duplicate and wasted effort. This result clearly demonstrates why there should be some form of cooperation between TCP and the 802.11 MAC in order to avoid such inefficiencies.

In summary, the results are surprising, especially because the distance of the end-device from the AP was just 10 metres or less, well within the typical signal range of an 802.11g infrastructure WLAN. Such end-user distances are not uncommon within home-office WLAN environments. Hence, TCP senders in the fixed Internet could be greatly affected by last-hop WLAN error conditions, more so than was previously assumed.

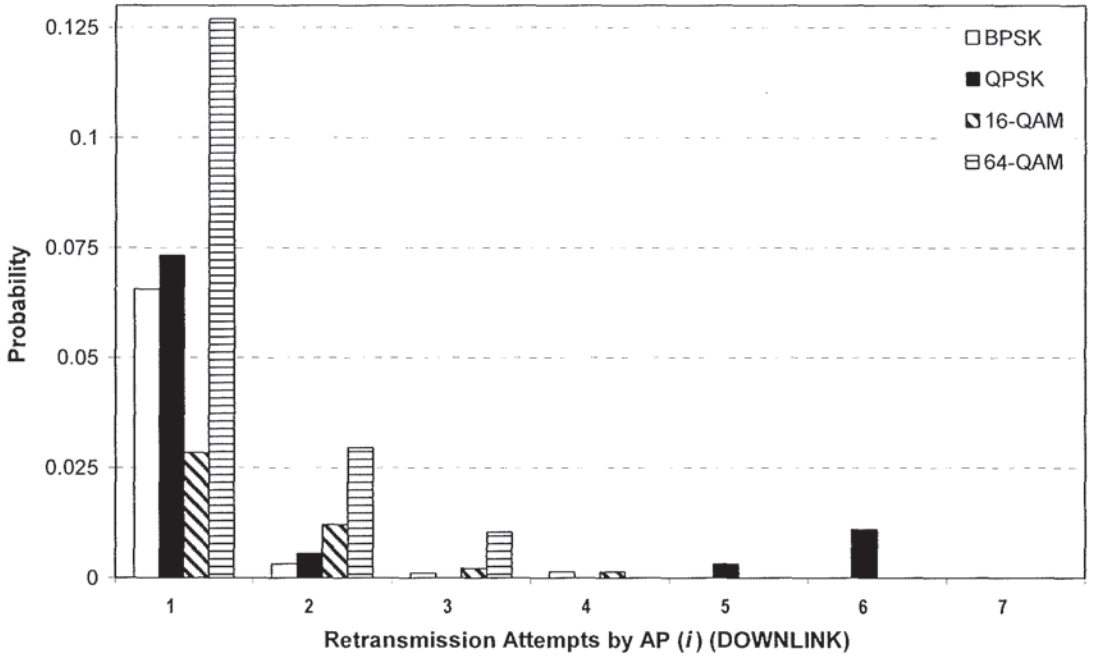


Figure 6.8: Probability distribution of frame retransmission attempts by AP (2m / SNR = 50dB)
(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

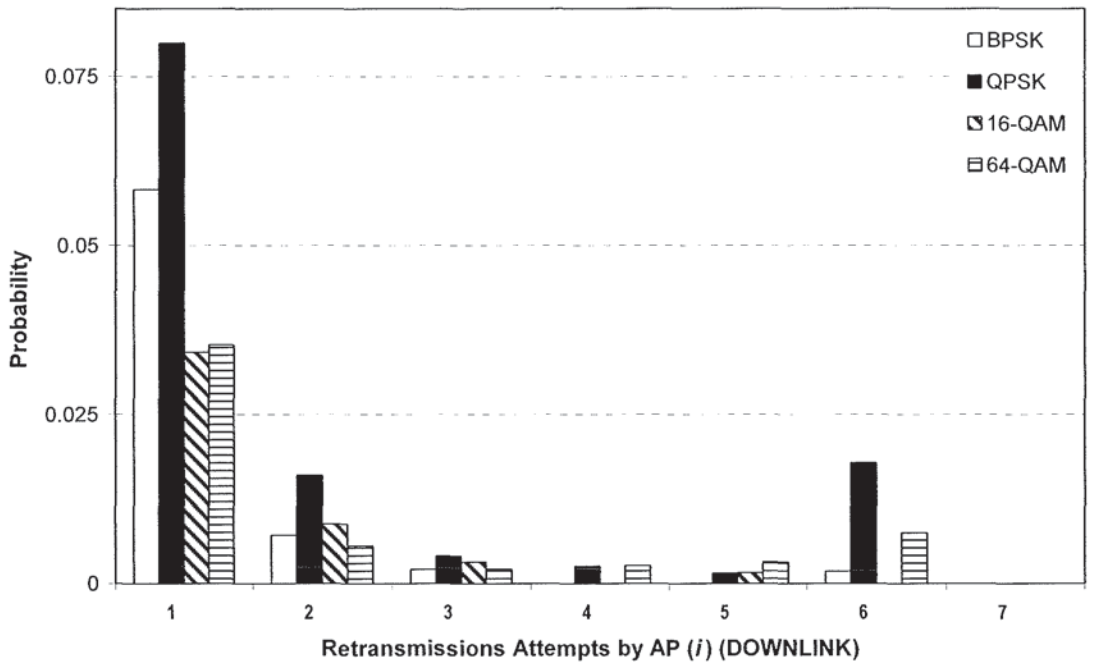


Figure 6.9: Probability distribution of frame retransmission attempts by AP (4m / SNR = 44dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

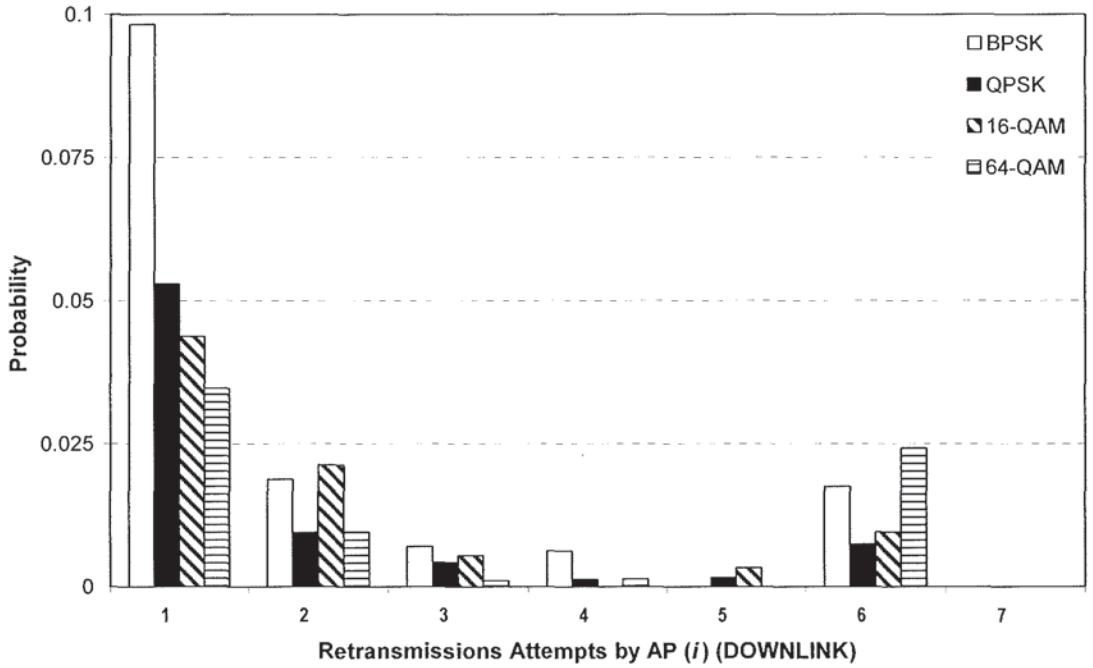


Figure 6.10: Probability distribution of frame retransmission attempts by AP (6m / SNR = 42dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

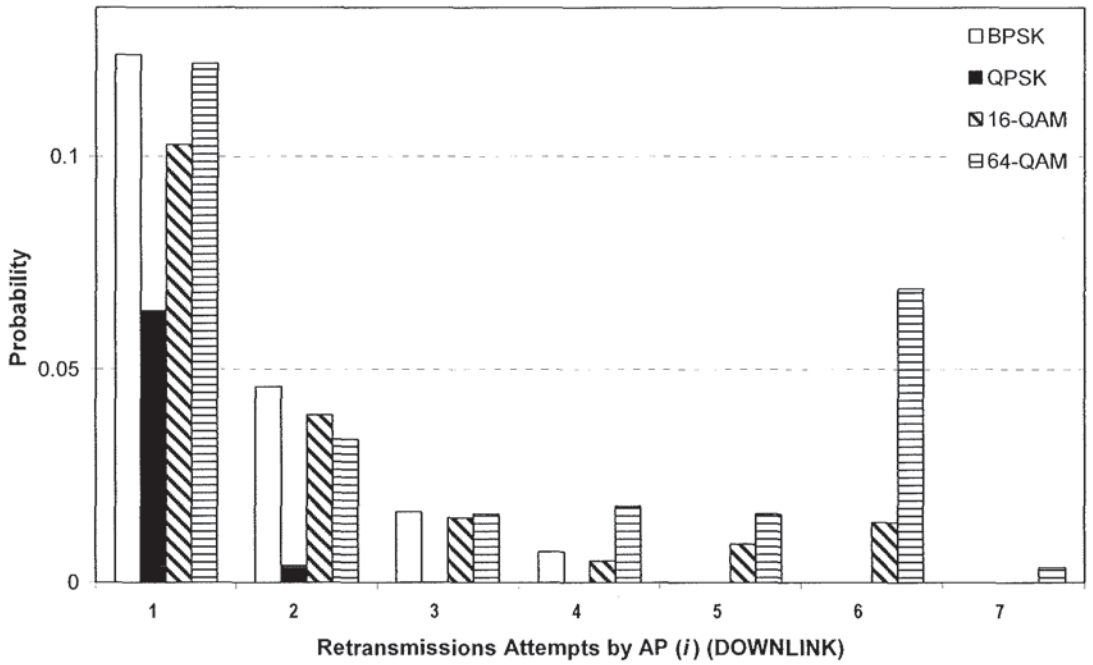


Figure 6.11: Probability distribution of frame retransmission attempts by AP (8m / SNR = 38dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

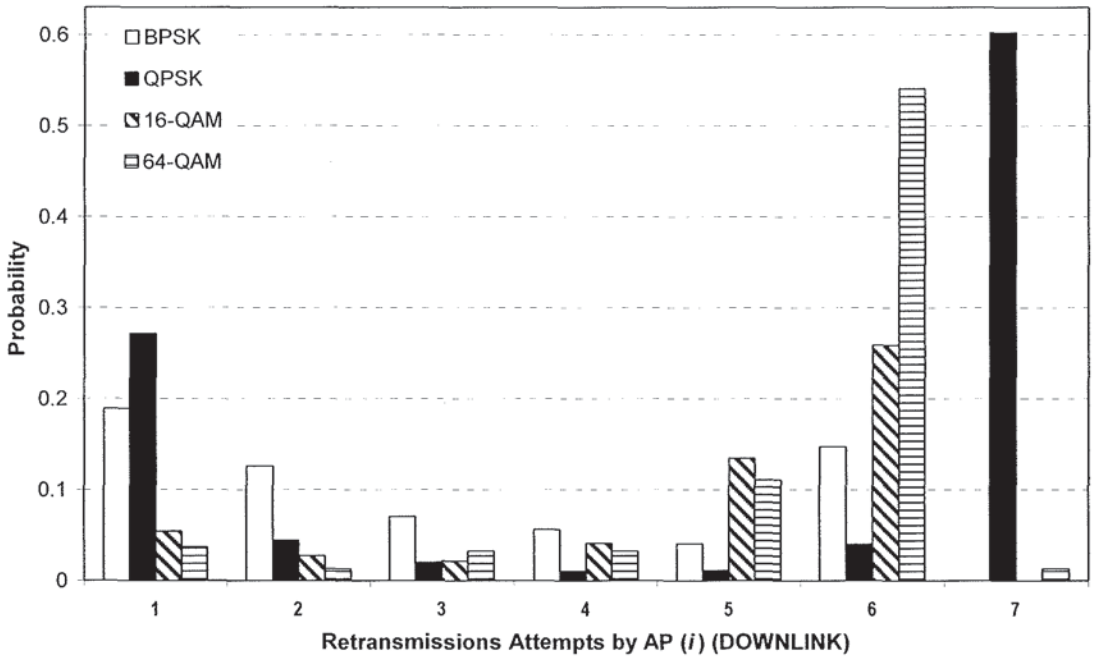


Figure 6.12: Probability distribution of frame retransmission attempts by AP (10m / SNR = 32dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

Probability Distribution of Frame Retransmission Attempts by End-Device

Figures 6.13 to 6.17 present the distributions of the uplink data frame retransmission attempt probabilities, $P_{UPLINK(i)}$, for each of the modulation schemes tested at distances of the end-device of 2, 4, 6, 8 and 10 metres from the AP respectively. Likewise, $P_{UPLINK(i)}$ is the probability of a data frame being retransmitted i times by the 802.11 end-device before success, as defined by Eq. 6.2. All probabilities have been computed plotted using the following calculation, $P_{UPLINK(i)} \times FER_{UPLINK}$. The retry limit, M, is equal to 15 for the end-device's MAC, which is the default setting as configured by the vendor of the IPW2200BG 802.11 adaptor. Again, the plots do not show the case where i is equal to zero, as technically these are not classed as retransmissions. These plots show the retransmission behaviour of TCP ACKs generated by the end-device's TCP-layer, encapsulated by 802.11 data frames.

Looking initially at Figures 6.13 (2 metres), all modulation schemes had incidences of frame retransmissions requiring up to four attempts, with 16-QAM retransmitting some frames 7 times before success. In Figure 6.14 (4 metres), a dramatic increase in the probabilities of i can be seen, with double, triple, and quadruple retransmission attempts dominating the probability space.

An interesting result occurred in Figure 6.15 (6 metres), where BPSK and QPSK had incidences of up to 15 retransmission attempts for frames. Note that such a high number of i would have resulted in significant delays for TCP ACKs in the transmit queue of the device's MAC [53].

At 10 metres from the AP, Figure 6.17 shows significant increases in the probabilities associated with retransmitting a particular data frame in the uplink direction for i ranging from 11 to 15. Note that a subset of the retransmissions for i equaling 15 may not have been successful either, thereby resulting in complete frame losses. Although 64-QAM possessed the highest probabilities, generally all modulation schemes followed a similar trend.

In this subsection it has been discovered that the MAC will exhaust its retry limit in the case of poor channel conditions, at the expense of increased delays for incoming

higher-layer data into the transmit queue. By having such a high value of M it causes unnecessary delays in returning TCP ACKs to the sender, who may actually timeout in the waiting process. If the delays are consistent, eventually the sender may inflate its RTO timer in-line with the inflated RTT, however an inflated RTT decreases the rate of growth of the *cwnd* size, leading to lack-lustre performance. An inflated RTO can also be an issue if a wireless end-user suddenly moves into a location with better signal quality, and in the event of an actual TCP loss the sender will take too long to respond.

It is therefore suggested that vendors of 802.11 client adaptors should strive to leave the default setting of the MAC retry limit to that specified in the original IEEE 802.11 standard [20].

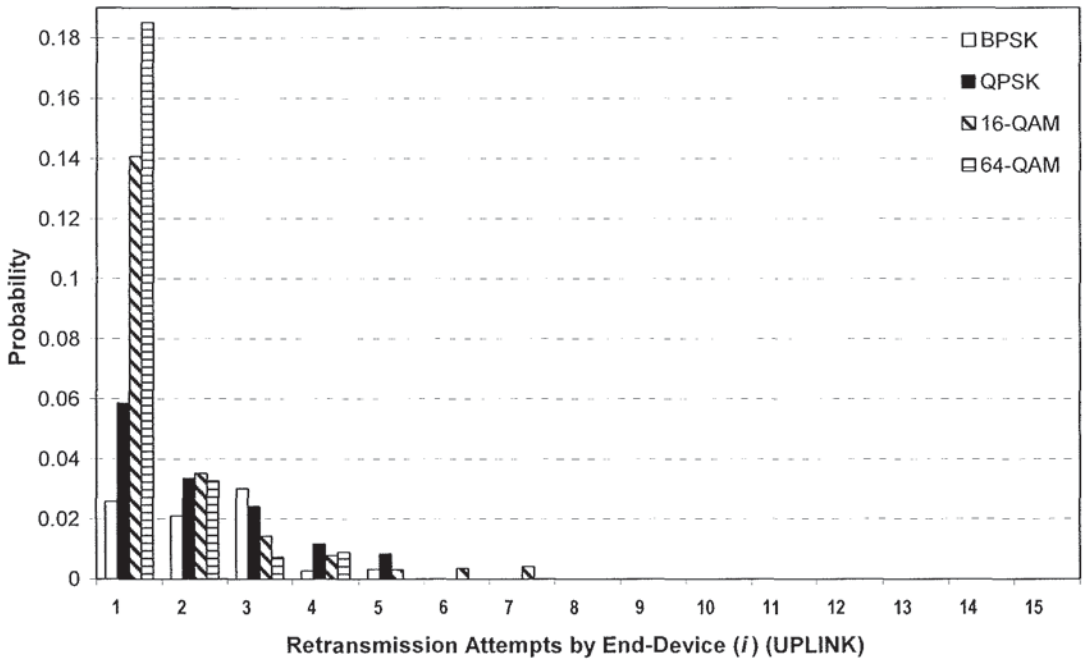


Figure 6.13: Probability distribution of retransmission attempts by end-device (2m / SNR = 50dB)
(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

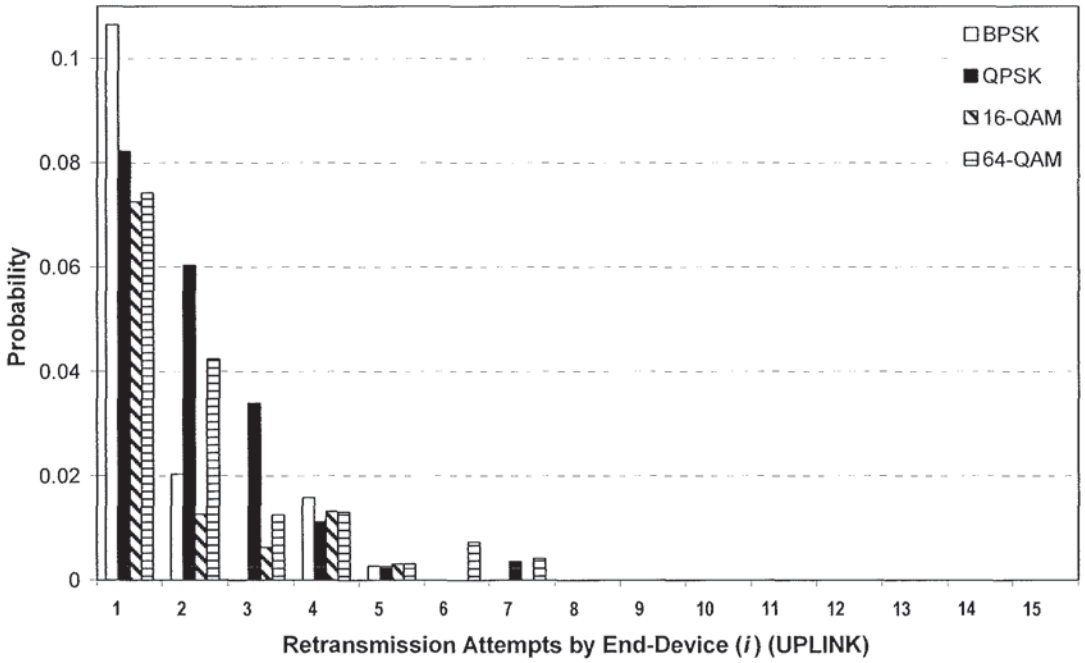


Figure 6.14: Probability distribution of retransmission attempts by end-device (4m / SNR = 44dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

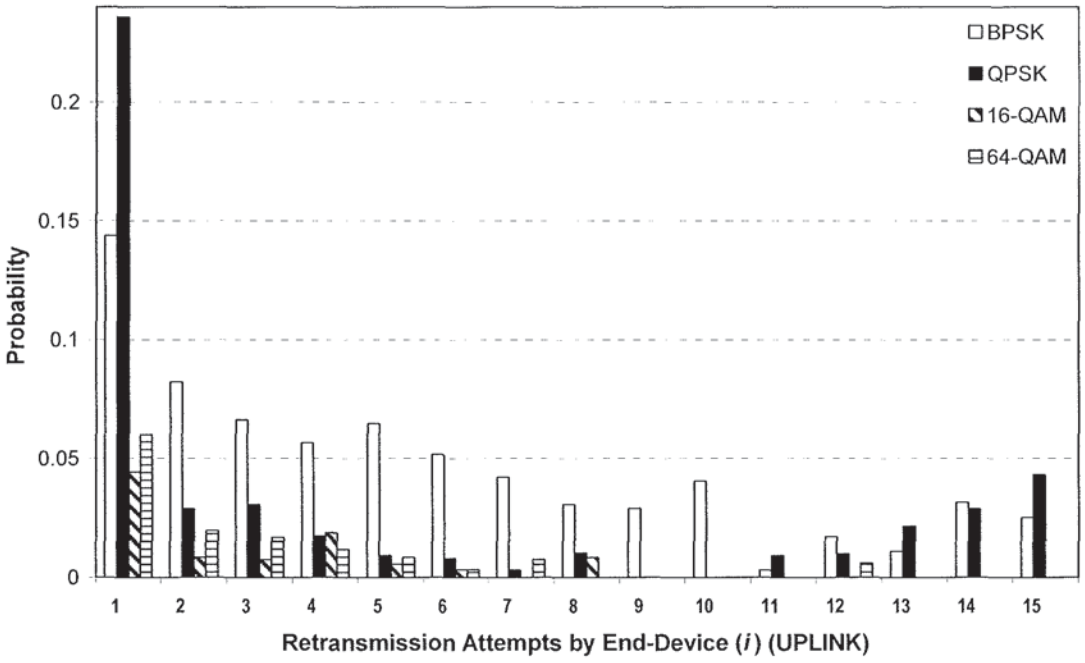


Figure 6.15: Probability distribution of retransmission attempts by end-device (6m / SNR = 42dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

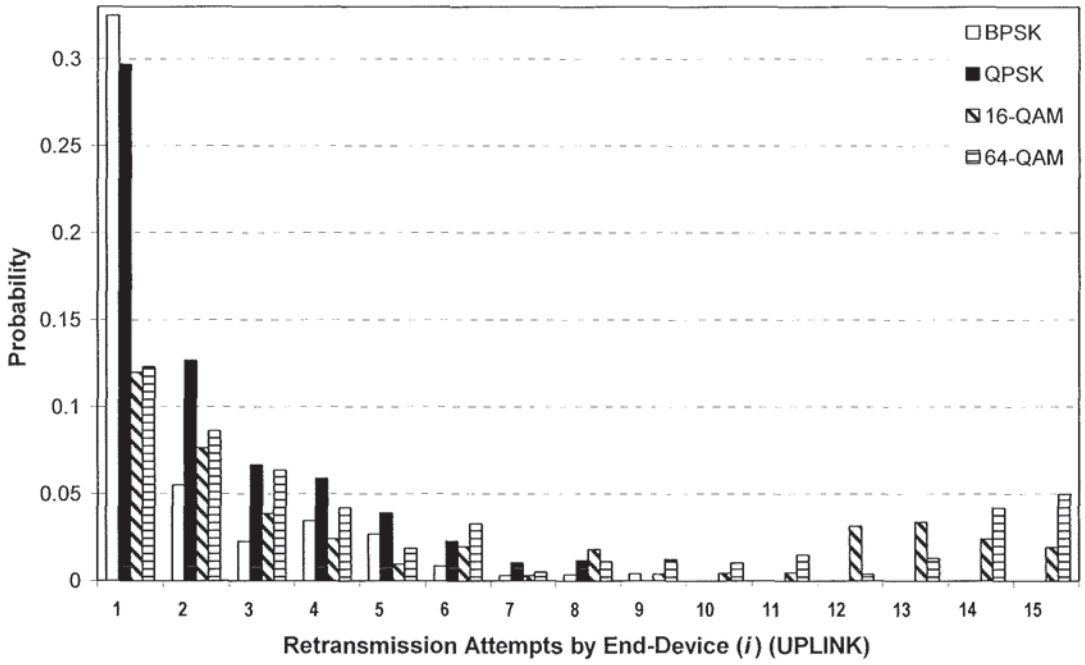


Figure 6.16: Probability distribution of retransmission attempts by end-device (8m / SNR = 38dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

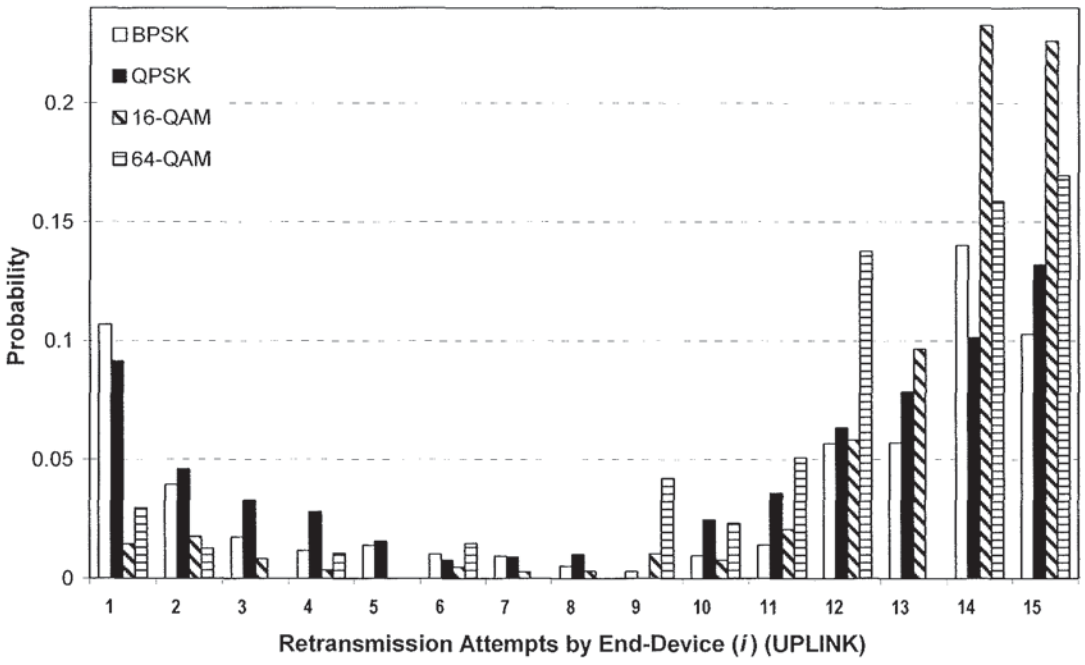


Figure 6.17: Probability distribution of retransmission attempts by end-device (10m / SNR = 32dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

6.5 Chapter Conclusions

In this chapter the objective was to provide new insights to researchers working in the area of TCP performance enhancements for wired-to-wireless paths; this is comprehensively conveyed with the proposal of a purpose-built testbed for conducting more advanced and accurate experiments using TCP over last-hop 802.11 WLANs. Strong reasons were also given for opting to conduct physical experimental studies over computer simulations alone.

A strong feature of the testbed is that it can be constructed from readily available hardware and software tools, encouraging researchers to add new dimensions to their experimental works. The importance of an Internet component within wired-to-wireless TCP experiments has also been stressed, being a key part of the testbed.

The uniqueness of the testbed is that it can be used to study the TCP-layer of the sending machine in the wired domain when there is an emulated Internet and a real-world wireless path in the journey of TCP segments and ACKs. In addition to studying TCP behaviour, the testbed allows researchers to conduct detailed studies into transmission behaviour of the IEEE 802.11 MAC. The novelty lies with the fact that it simultaneously allows the capturing of data/information from the TCP-layer and from over the 802.11 WLAN from all angles of the same TCP experiment, from both the sender's and the receiver's perspective. Having such a wealth of captured data becomes invaluable when evaluating the end-to-end performance of TCP over wired-to-wireless paths.

In the next chapter the testbed will include an investigation of TCP sender-side performance with a variable number of last-hop 802.11 devices in order to understand TCP behaviour as a function of WLAN channel contentions.

To highlight and demonstrate its effectiveness, some TCP experiments were conducted over the testbed using a real-world 802.11 WLAN that is reflective of typical home-office indoor environments, using a best-case scenario. A wide set of results from the testbed confirm that frame errors are prevalent over WLAN channels in both downlink

and uplink channels. But interestingly, it was also discovered that the reverse channel of the WLAN possessed its own frame error behaviours, quite different from forward channel error rates. Further observations led to the conclusion that there may be more correlation between reverse channel error rates over the WLAN and the number of times a TCP sender invokes its retransmission mechanism, more so than for forward channel FERs.

Also investigated in this chapter were the probability distributions of the number of times a particular 802.11 data frame was retransmitted by the AP and by the 802.11 end-device; this again highlighted the depth of data that can be extracted from the testbed. The results demonstrated that when channel conditions degrade, an 802.11 AP and end-device will maximise the use of their MAC ARQ mechanism, right up to the configured retry limit, *M*. In fact, at end-device distances of just 10 metres from the AP, *M* and *(M-1)* retransmissions of the same data frame are more likely to occur than single or double retransmissions. Such real-world observations may be useful to those working with error correction techniques such as erasure-control coding to design optimal codes, or equally to those wishing to gain an understanding of how TCP's RTO timer could be closely aligned to the highly variable delay conditions occurring over an 802.11 WLAN due to localised retransmissions.

Chapter 7

Evaluation of Linux Sender-Side TCP Implementations

7.1 Introduction and Motivations

The legacy TCP congestion control (AIMD) algorithms, referred to as Tahoe and Reno, have been very successful in making the Internet function efficiently to date. They have provided a reliable connection-oriented end-to-end data delivery service to end-users across the globe. In recent years however, there appears to be a changing environment, in which end-users of TCP are no longer restricted by cables [15]. The advent of the IEEE 802.11 WLAN technology has allowed end-users of TCP applications to detach from the traditional methods of accessing services from the fixed Internet, so they can be mobile and work location independently from within their home/office environments. Specifically, many homes and offices are now fully equipped with 802.11 WLANs, enabling users and devices to connect to the Internet via a local gateway AP in order to access services that still run over TCP for reliable end-to-end transportation [135]. It is therefore a common scenario today for TCP segments originating from a server somewhere in the Internet to traverse a wireless link in the last-hop portion of the end-to-end journey [295].

In summary, the key problems for a sender-side TCP of having an 802.11 WLAN at the last-hop are: i) higher BERs from the natural effects of the radio transmission medium, leading to higher FERs and losses [11], ii) the prevalence of over-the-air collisions of frames when there are multiple 802.11 end-devices generating uplink and downlink TCP flows via the AP, and, last but not least, iii) highly variable transmission delays of frames over the WLAN in both uplink and downlink directions [296], which is due to medium access delays by all sending 802.11 devices (including the AP), as well as sudden queuing delays when the 802.11 MAC stop-and-wait ARQ mechanism is in progress.

In this chapter, detailed evaluations are presented for some of the most recently proposed sender-side TCP enhancements when used over wired-to-wireless conditions,

in an attempt to observe how they weigh up against the legacy TCP protocol, Reno. The problems faced by the TCP Reno over wireless paths have been pursued by many researchers in an attempt to increase its sending efficiency. The idea supported throughout this thesis is to make enhancements to the protocol at the sending side in the fixed domain, which can be more easily rolled out across servers in the Internet. Also, by keeping modifications only to the protocol's code, the rules of the OSI layering principle for network stacks are obeyed [3].

A common solution to the problem that has been suggested and proposed by many authors is to customise the AIMD mechanisms of TCP Reno in order to better predict network conditions and make sound judgements on how to react when a segment loss occurs in the wired-to-wireless path, or when there is a random delay in the returning of ACKs [13]. Obviously the pinnacle of such research would be for a TCP sender to know exactly the cause of the loss or delay. If the loss occurs in the wired portion, which would indicate network congestion, then TCP should adjust its sending rate to alleviate a bottleneck. If the loss occurs in the wireless portion, which would imply an erroneous radio channel, then there is no need for TCP to worry about reducing its sending rate as the AP will usually continue sending incoming data from its transmit queue at the same data rate over the WLAN.

The focus therefore in this chapter is on some of the popular and recently proposed enhancements to the standard TCP's AIMD algorithms for wired-to-wireless paths, which have also made their way into the Linux v2.6 kernel's TCP stack [82]. Research within the area suggests that TCP Hybla [208], TCP Veno [199], and TCP Westwood+ [195] appear to be some of the most popular and recent candidates for enhanced performance over wired-to-wireless conditions that are also available in the Linux v2.6 kernel. To increase diversity, TCP CUBIC [286] is also put under test, as it is default algorithm in Linux kernels since v2.6.19. The rationale here is that Linux web-servers are widely deployed across the Internet, and server administrators may not make changes to the default TCP congestion algorithm, implying that last-hop wireless end-users around the world are more likely to be served by TCP CUBIC when accessing web-services from such servers. Finally, legacy TCP Reno is also put under test, acting as a base-line in order to make comparisons against and evaluate the enhanced AIMD algorithms.

Due to the amount of work that has been undertaken in the area of TCP performance over wired-to-wireless paths, a real need has arisen for the screening of proposals in order to identify suitable candidates for future progression. Evaluating the performance of TCP over wireless paths is not a straight forward task, and many studies opt for simulation or emulation in order to study TCP behaviour in a non-complex manner. Several studies try to add realism through the use of an experimental testbed, consisting of a real-world last-hop wireless network. However researchers only tend to use legacy TCP implementations or one which they feel is adequate, putting little emphasis on the actual comparison of sender-side TCP variants, but rather on the core performance of TCP. Finally, very little literature was found that undertakes a systematic evaluation of real-world TCP AIMD implementations in real-world typical conditions, reflecting a true wired-to-wireless path for TCP. Therefore, concrete conclusions relating to the merits of competing wired-to-wireless proposals in the real-world have been difficult to make based on currently available published results.

The aim in this chapter is to systematically use the wired-to-wireless testbed proposed in Chapter 6 to compare and evaluate the performance of competing TCP AIMD enhancement proposals for tackling the issues related to wired-to-wireless conditions. It is important to emphasise on the outset that the goal is not to achieve exhaustive testing of each proposal, but rather to perform an initial screening of them. Specifically, a set of benchmark tests over real-world conditions are performed, reflecting a typical home-office 802.11 WLAN environment that is connected to a wired backbone, using the Linux v2.6.20 kernel as the TCP server machine. Experimental results of the performance of TCP Reno, CUBIC, Hybla, Veno, and Westwood+ are presented from tests over a range of typical real-world scenarios involving multiple 802.11 end-devices in the last-hop WLAN.

7.2 Prerequisite Knowledge

Before proceeding with the experimental descriptions and methods, the basic functionalities of the following sender-side TCP AIMD variants are briefly reviewed: Reno, CUBIC, Hybla, Veno, and Westwood+. Each of these algorithms has been the

subject of considerable interest and testing in recent years, with patches of their specific implementations publicly available in the Linux v2.6 kernel patches.

It is assumed that the reader is somewhat familiar with the basic principles of congestion control in TCP, i.e. the concept of the sender's *cwnd*, and its relationship with the AIMD mechanisms. It should also be stated that a TCP sender performs loss detection in either of the two ways [99]: 1) via a RTO timer expiry event whilst waiting for an ACK from the receiver, and 2) through the arrival of three DUPACKs for the same unacknowledged data. Because TCP's error detection and recovery is performed end-to-end, it also operates transparently for segment losses occurring over the wireless path of a connection. For more detailed insights into the functionality of each of the TCP AIMD algorithms in this section the reader is referred to the respective original literature.

TCP Reno

TCP Reno was introduced as an enhancement to the original TCP Tahoe [1] by altering the way in which it reacted to segment losses due to congestion. It introduced the fast recovery algorithm, which is activated immediately after the fast retransmit algorithm upon the arrival of three DUPACKs. Although Reno was never designed with the intention for usage over wireless paths, it was very widely adopted by the Internet community and is still likely to be in use today by legacy servers serving last-hop wireless end-users. Reno's AIMD algorithm in more detail is as follows:

upon every ACK:

```
if cwnd < ssthresh
      cwnd ← cwnd + 1

if cwnd > ssthresh
      cwnd ← cwnd + 1/cwnd
```

upon every Loss:

RTO

```
      ssthresh ← cwnd/2
      cwnd ← 1
```

3×DUPACKs

```
ssthresh ← cwnd/2
cwnd ← ssthresh + 3×MSS
foreach DUPACK:
    cwnd ← cwnd + MSS
upon new ACK:
    cwnd ← ssthresh
```

It is the above rules that govern Reno's behaviour in light of segment losses and network congestion events. The *cwnd* size dictates how much data can be pushed into the network at any one time, effectively controlling the sending rate. The rate of incoming ACKs returning from the recipient governs the additive-increase/multiplicative-decrease mechanism of the *cwnd* at the sender. The *ssthresh* variable keeps a note of the current end-to-end network capacity, in an attempt to prevent further segment losses from occurring due to the injection of too much data into the network. As can be seen, Reno was designed with network congestion in mind, where losses were presumed to occur mainly due to buffer overflows along connection paths, and not due to random wireless channel errors.

TCP CUBIC

TCP CUBIC is a sender-side congestion control algorithm that was designed to meet the demands of today's high-speed long distance networks, with the core objective being to take advantage of unused end-to-end bandwidth, a problem that is exhibited by standard TCP algorithms due to their poor AIMD response times. In essence, CUBIC uses a *cubic function* to govern the size of its *cwnd*. A variable W_{max} is maintained as the size of the *cwnd* prior to a reduction event. When the *cwnd* size increases again, it grows quickly initially, but decelerates this growth as its size approaches W_{max}, accelerating again as its size continues to grow beyond W_{max}. In CUBIC the *cwnd* evolves according to the following rules:

upon every ACK:

```
cwnd ← cwnd + [C(T - K)^3 + Wmax]
```

upon every Loss:

RTO or **3×DUPACKs**

```
Wmax ← cwnd
T ← 0
cwnd ← cwnd + [C(T-K)^3 + Wmax]
```

where:

$$K = \sqrt[3]{\frac{W_{max} \times \beta}{C}}$$

and C is a scaling constant equal to 0.4, T is the time elapsed (in milliseconds) since the last *cwnd* reduction (or segment loss event), and β is the multiplicative decrease factor after a segment loss occurs, equal to 0.2, regardless of cause.

TCP Hybla

TCP Hybla was proposed with the primary objective of providing better performance in end-to-end paths that possess larger segment RTTs due to wireless hops such as a satellite radio link. Briefly, its core enhancements include a modification of the standard TCP AIMD algorithm for higher latency paths, enforcing the use of the SACK policy (to tackle multiple losses due to larger *cwnd* sizes), the use of segment timestamps (to tackle RTO timer exponential back-off issues due to higher RTTs), and the usage of a proprietary segment 'pacing' technique.

Hybla's modifications to the *cwnd* evolution are a direct result of analytical studies on its behaviour in such conditions, leading to the conclusion that altering the *cwnd*'s sole dependence on the RTT for growth is a viable solution. In essence, the higher the latency of a segment's journey, the larger the *cwnd* size needs to be in order to achieve a given throughput, generally calculated by *cwnd*/RTT. This leads to the rationale that for higher latency TCP connections the *cwnd* growth should be more accelerated to take advantage of end-to-end system bandwidth more effectively. In more detail, Hybla uses the notion of a parameter ρ as an equalisation term for the RTT_0 of a reference wired TCP connection, and the actual *RTT* of high latency connection. Hybla then equalises the rate of injection of new data (i.e. how the *cwnd* grows) by replacing the standard TCP AIMD algorithm with the following procedures:

upon every ACK:

```
if cwnd < ssthresh
      cwnd ← cwnd + 2^ρ - 1
if cwnd > ssthresh
      cwnd ← cwnd + ρ²/cwnd
```

where:

$$\rho = \frac{RTT}{RTT_0}$$

upon every Loss:

RTO

```
ssthresh ← cwnd/2
cwnd ← 1
```

3×DUPACKs

```
ssthresh ←  max(FS/2, 2×MSS)
cwnd ← ssthresh + 3×MSS
foreach DUPACK:
      cwnd ← cwnd + MSS
upon new ACK:
      cwnd ← min(ssthresh, FS)
```

TCP Westwood+

TCP Westwood+ is a direct modification to the standard TCP AIMD algorithm in an attempt to provide enhanced performance over wired-to-wireless paths. Westwood+ uses an end-to-end *bandwidth estimator* (BWE) to alter the size of its *cwnd* and the *ssthresh* value, specifically after a segment loss event. Using a 'low-pass filtering' mechanism, it monitors the rate of returning ACKs from the recipient. The idea behind this is to continuously be aware of the end-to-end bandwidth of the connection, so that at the point of a segment loss occurring Westwood+ is able to adaptively set the *ssthresh* and *cwnd* to suitable values which better reflect network conditions. Its authors confirm specifically that Westwood+ can increase throughput over wireless networks.

The bandwidth estimation in Westwood+ is performed by counting and filtering the flow of incoming ACKs. For every RTT, an estimated bandwidth sample, b_k, is obtained via: $b_k = d_k/\Delta_k$, where d_k is the amount of data acknowledged during the last RTT, Δ_k. In more detail, the following rules define the AIMD mechanism of Westwood+:

upon every ACK:

```
if cwnd < ssthresh
      cwnd ← cwnd + 1
if cwnd > ssthresh
      cwnd ← cwnd + 1/cwnd
compute end-to-end BWE
```

upon every Loss:

RTO

```
ssthresh ← max(2, (BWE×RTT_min)/MSS)
cwnd ← 1
```

3×DUPACKs

```
ssthresh ← max(2, (BWE×RTT_min)/MSS)
cwnd ← ssthresh
```

where RTT_{min} is the minimum of all the measured RTT samples.

TCP Veno

TCP Veno is a direct modification to the standard TCP AIMD algorithm, with the objective of trying to improve performance over wired-to-wireless paths by heuristically being able to discriminate between segment losses due to congestion in the wired network and random losses due to the wireless radio link in the end-to-end path. The key feature of Veno is that it monitors the network path for levels of congestion, and then uses this information to determine the cause of loss. If a loss occurs whilst Veno is in the congestive state, it assumes that the loss is due to network congestion; otherwise it assumes that the loss is random. In each of these situations Veno adjusts the *cwnd* size differently (i.e. the multiplicative decrease phase), taking

into account that random losses require a less aggressive reduction of the sender's *cwnd* size in order to maintain a higher throughput over the wireless portion of the connection. Another feature of Veno is its modified linear portion of the additive increase algorithm of standard TCP, so that when Veno is in the congestive state the *cwnd* growth is less aggressive. This effectively allows any self-induced network congestion to be relieved without segments being dropped in the wired network, helping Veno connections to maintain a higher overall throughput by remaining in an optimal operating region for longer.

Veno is able to determine whether or not it is in the congestive state by using calculations adopted from TCP Vegas [193] to calculate the value of N, the number of backlogged segments at a bottleneck queue somewhere along the connection path. The value of N is continuously updated during a live connection, and is derived from: $N = (ExpectedRate - ActualRate) \times RTT_{min}$, where RTT_{min} is the minimum of all the measured RTT samples. Hence, if $N < \beta$ when a segment loss occurs then Veno assumes that the loss is more likely to be random (most likely occurring in the wireless link), than due to congestion. The authors in [199] suggest a value of 3 for β from their experimental work. In more detail, below is the AIMD algorithm of Veno:

upon every ACK:

```
if cwnd < ssthresh
      cwnd ← cwnd + 1
if cwnd > ssthresh
      if N < β
            cwnd ← cwnd + 1/cwnd
      if N ≥ β
            for (every other ACK)
               cwnd ← cwnd + 1/cwnd
```

upon every Loss:

RTO

```
      ssthresh ← cwnd/2
      cwnd ← 1
```

3×DUPACKs

```
if N < β
        ssthresh ← cwndx(4/5)
else
        ssthresh ← cwnd/2

cwnd ← ssthresh + 3×MSS
foreach DUPACK:
        cwnd ← cwnd + MSS
upon new ACK:
        cwnd ← ssthresh
```

It can be noticed from the above pseudo-code that when the value of N is less than β then the reduction of the *cwnd* is less aggressive (i.e. only a 20% reduction in its size). The authors' experimental work in [199] demonstrates that any reduction factor for the *cwnd* of less than 50% works well, with reductions of less than 25% being the most desirable. Note also that when N is greater than or equal to β, the reduction of the *cwnd* is identical to that of TCP Reno, so is Veno's *slow start* phase and its *congestion avoidance* phase when N is less than β.

7.3 Testbed Setup and Procedures

All experiments were conducted using an extended version of the wired-to-wireless testbed proposed in Chapter 6. Figure 7.1 illustrates the newly adopted testbed, the details of which are given in the following subsections. The extension of the testbed refers to the fact that it now includes multiple 802.11 end-devices (as opposed to just a single end-device in Chapter 6), and a live connection to the Internet backbone for all end-devices.

7.3.1 Hardware Setup

As can be seen from Figure 7.1, the main hardware change is the incorporation of a gateway into the live Internet. The idea behind this is to allow some 802.11 devices in the last-hop WLAN to access real web-services, creating contention over the WLAN

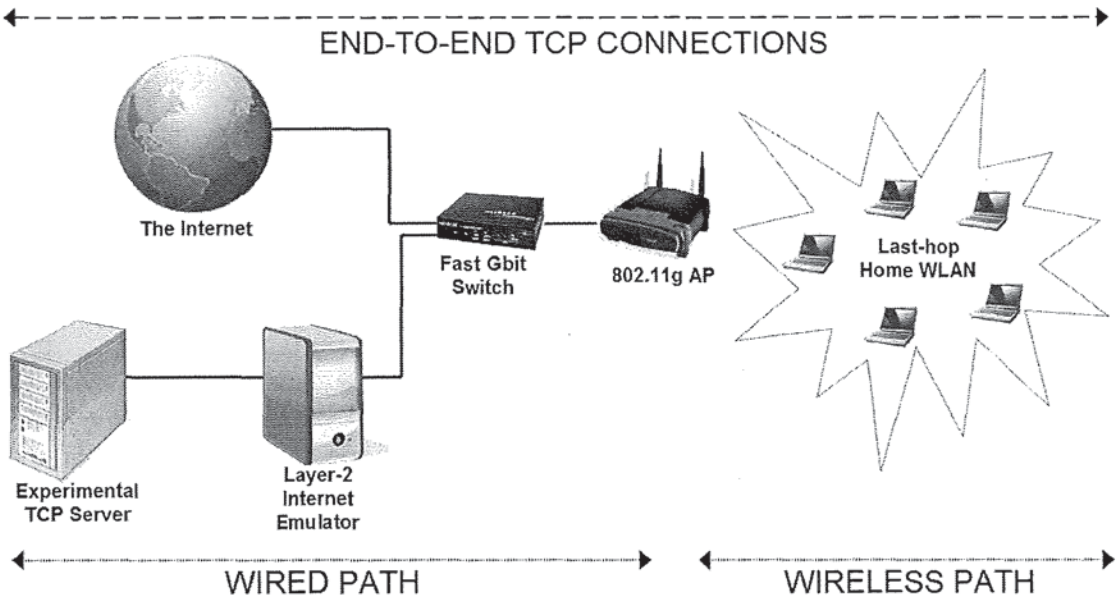


Figure 7.1: The adapted wired-to-wireless testbed

and generating additional downlink/uplink traffic. This adapted topology ensures that an end-device in the WLAN should not be able to discriminate between TCP data being sent from a server somewhere in the live Internet and TCP data being sent from the experimental testbed server locally. The basic idea behind having the experimental TCP server coupled with the Internet emulator is simply to create a lower-level of abstraction of a typical server in the Internet. This technique ensures the server isn't in a geographically distant location and can actually be controlled, and whose TCP-layer can be observed and measured. It can be appreciated that in the live Internet it is not practical to instrument or measure the TCP behaviour of a web-server. The view on the work of this chapter is that the bulk of TCP data sending is done by web-servers in the Internet, and hence it is more beneficial to study TCP settings and behaviour at the server in the wired domain.

The IEEE 802.11g AP settings were left untouched, set to their factory defaults. The last-hop WLAN consists of a maximum of five 802.11g end-devices, which is in contrast to the configuration in Chapter 6.

Client Name	Description	OS	Memory	CPU	802.11g Interface
SEAMOSS *(REFERENCE END-DEVICE)*	HPCompaq tc4200	Windows XP SP2	512MB	1.8GHz	Intel IPW2200BG
HANALENI	HPCompaq nc6220	Linux v2.6.22 FC6	512MB	1.9GHz	Intel IPW2100BG
TEMPLETREE	Apple iPod Touch 8GB	iPhone OS	128MB	412MHz	Marvell W8686B22
CEMPAKA	DELL Inspiron 1300	Windows XP SP2	256MB	1.5GHz	Intel IPW2200BG
DAVANA	Compaq Presario M2000	Windows XP SP2	512MB	1.5GHz	Intel IPW2200BG

Table 7.1: WLAN end-device names and specifications

The WLAN was configured using DCF in an infrastructure configuration. Details of their assigned names and specifications have been provided in Table 7.1. Note that SEAMOSS is the reference end-device for all experiments in this chapter i.e. all focus and measurements were based on this end-device.

Finally, a fast Netgear SR2016 gigabit switch was used to interconnect both wired backbones to the AP. All wired connections between the experimental server, the live Internet, the Internet emulator, the switch, and the AP were using full-duplex 10/100Mbps Ethernet cables to eliminate the need for additional hardware.

7.3.2 Software Configurations

The experimental TCP server in the testbed was updated to run the slightly newer Linux v2.6.20 kernel, which was also instrumented with extensions from the web100 project. Each of the TCP congestion control algorithms to be studied were built into the kernel upon compilation in order to provide consistency and control over the differences in implementations that exist for differing kernel versions.

At the TCP-layer on the server the following options were enabled: SACK, window scaling, delayed-ACKs, the nagle algorithm, and path MTU discovery. Such settings are typical of most TCP implementations today [108]. The TCP MSS was set to 1460 bytes. Note that it is up to the contributed TCP congestion control patches in the Linux kernel as to which of the above options are used, which is of course beyond the control and the scope of this chapter. Also, it is not uncommon for certain TCP patches to make alterations to some of the standard options, either for greater processing efficiency, or for their usage in non-conventional ways as part of the enhancement.

The iperf tool was used throughout the testing to transfer fixed amounts of TCP data from the server to a designated 802.11g client (SEAMOSS) in the last-hop WLAN. In order to minimise the effects of local host queues and flow interactions, and unless otherwise stated, only single connection TCP flows were run from the experimental server per iperf run.

As well as using web100 data at the server, the tcpdump tool was also used at the TCP layer on the experimental server and on SEAMOSS to capture all sent and received segments and ACKs per experimental trial; the dump files were then analysed using the popular tcptrace tool.

As listed in Table 7.1, all 802.11g devices were running default TCP/IP networking settings of the installed operating system in order to maintain realism throughout. This is typical of most 802.11 WLAN end-users, who generally do not want to be altering system configurations, due to either a lack of understanding or not having an interest in them. They simply want to power up the device, let the operating system boot, and then experience wireless connectivity to the Internet.

7.4 Real-World Home WLAN Scenarios

The infrastructure WLAN component of the testbed was setup and configured to represent a modern-day home WLAN environment, consisting of up to five wireless end-users (802.11g devices) connected to the Internet via a local gateway, and the 802.11g AP. In this context it is assumed that this represents a typical US family or household consisting of two adults and three youths [135]. Each family member presumably owns an 802.11g compatible wireless device, and is connected to the AP with the ability to access services and unlimited content from the Internet. For the purposes of the study, five different physical locations are allocated within an actual home where wireless users typically could be situated, each independently downloading and uploading data to and from the Internet via the AP. At each of the chosen locations a particular 802.11g device from Table 7.1 is permanently assigned. The layout of the home environment is illustrated in Figure 7.2 (not to scale), clearly showing the positions of the five chosen locations (marked 1 to 5), as well as the location of the AP. The dimensions of the home are 12.3 metres by 10.4 metres, with internal room wall partitions made from two layers of plasterboard with air cavities between them.

To create as much realism as possible two key scenarios within the home WLAN will be investigated in order to observe how the TCP sender-side enhancements in the wired domain perform over a variety of wireless conditions in the last-hop journey of segments. The first scenario investigates the impacts on the TCP sender of multiple 802.11 end-users in the WLAN. The second scenario investigates the impacts on the TCP sender of varying the locations of particular end-user (i.e. varying the SNR) in the home WLAN.

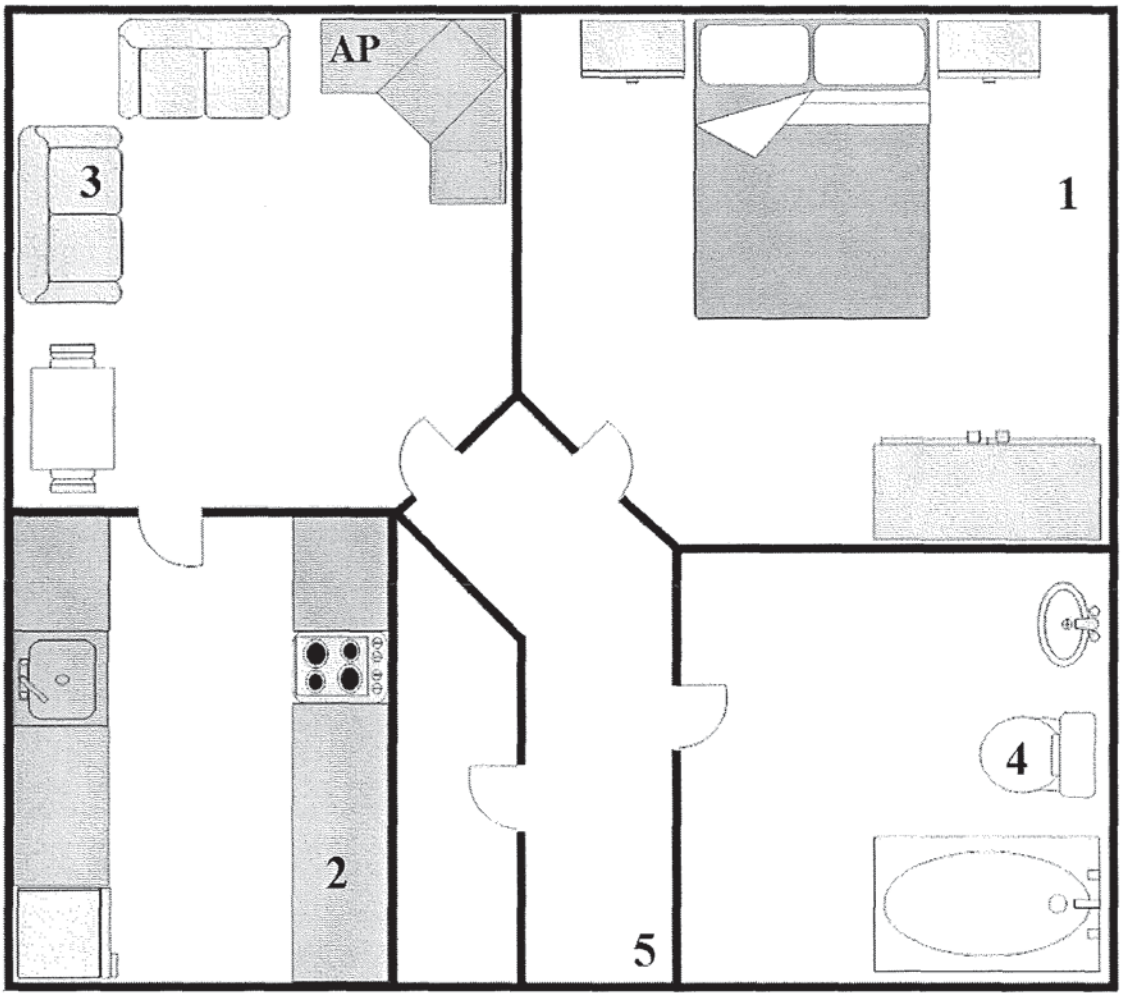


Figure 7.2: The real-world home WLAN floor plan (12.3 x 10.4 metres)

7.4.1 Scenario 1 – Multiple 802.11g End-Users

In this section five sub-scenarios, ranging from A to E, are considered. In sub-scenario A there is just a single end-user in the WLAN, positioned at location 1. The name of this device is SEAMOSS. In Table 7.2 each of the sub-scenarios are described in more detail, providing information on where the devices were positioned, and the actions that were performed by each.

Sub-Scenario	Client Name	Client Action	Location
A (SNR=32dB)	SEAMOSS	Downloading iPerf TCP Data	1
B (SNR=28dB)	SEAMOSS	Downloading iPerf TCP Data	1
	HANALENI	Downloading FTP Data	2
C (SNR=26dB)	SEAMOSS	Downloading iPerf TCP Data	1
	HANALENI	Downloading FTP Data	2
	TEMPLETREE	Streaming Video from BBC iPlayer	3
D (SNR=23dB)	SEAMOSS	Downloading iPerf TCP Data	1
	HANALENI	Downloading FTP Data	2
	TEMPLETREE	Streaming Video from BBC iPlayer	3
	CEMPAKA	Uploading FTP Data	4
E (SNR=25dB)	SEAMOSS	Downloading iPerf TCP Data	1
	HANALENI	Downloading FTP Data	2
	TEMPLETREE	Streaming Video from BBC iPlayer	3
	CEMPAKA	Uploading FTP Data	4
	DAVANA	Streaming Video from BBC iPlayer	5

Table 7.2: Sub-Scenarios A to E

In summary then, each sub-scenario from A to E increases the number of 802.11g devices in the WLAN, from 1 to 5 respectively. Note that for each sub-scenario, and in order to maintain consistency with the results, SEAMOSS at location 1 is always the subject device as the recipient of iperf data from the experimental TCP server, i.e. it is always the TCP client in all tests. All other devices were communicating with the live Internet as per Table 7.2, performing download and upload operations with real-world data. Note that the 802.11g WLAN was in saturation mode for each sub-scenario, i.e. all devices were downloading or uploading data for the full duration of the TCP data download by SEAMOSS.

7.4.1.1 Experiments and Selected Measurements

For each of the sub-scenarios A to E, experiments consisting of *small*, *medium*, and *large* TCP data flows were performed in order to cover a diverse range of testing styles. The the *small* flows transfer 1 Mb of data (representative of web-page downloads including all objects via HTTP v1.1), the *medium* flows transfer 10 Mb of data (representative of downloading, for example, hi-resolution image files via HTTP v1.1), and the *large* flows transfer 50 Mb of data (representative of downloading, for example, an open-source application package via FTP). All data flows are from the experimental TCP server to SEAMOSS in the last-hop WLAN in a downlink fashion. All TCP segments (including TCP ACKs returning from SEAMOSS) passed through the Internet emulator machine. Each flow is therefore representative of an 802.11g end-user within a home environment downloading a file using TCP from a web-server located in the Internet.

The small transfers test the initial start-up behaviour of the various TCP congestion control algorithms, where it is mainly the slow start algorithm that governs the performance for this type of flow [49], as these connections barely reach a steady bandwidth probing state. In contrast, the medium and large transfers test the steady-state behaviour of the various algorithms, which rely mainly on the congestion avoidance algorithm for the bulk of the data transfers. To recall, the congestion avoidance phase has been designed to gently and fairly probe for maximum end-to-end bandwidth, until a segment loss occurs somewhere in the path.

In order to obtain a good performance representation of the variability between runs, each of the small, medium, and large data flows was repeated three times for each TCP algorithm per sub-scenario in the home WLAN. Hence all results are the averages (arithmetic mean) of all measurements made.

The key performance measurements that were of interest from experiments were a) the *time taken to transfer* the entire amount of data per flow, and b) the *average throughput achieved* during the lifetime of a flow, which are both correlated. These are the most subjective of performance indicators as far as an 802.11g end-user in the WLAN is concerned, and hence these are the only results presented initially. It is appreciated that there are many additional indicators of TCP performance; however the idea in this chapter is simply to evaluate how the various proposals perform in a real-world context, where wireless end-users are expectant of high performance from the Internet, i.e. quick downloads and response times.

As a supplement, a results subsection has been included that presents the *average connection RTO value*, which was obtained by computing the median of all the individual web100 samples that were collected from the running Linux kernel on the TCP server during each experimental run.

TCP uses the RTO timer [107] to ensure data delivery in the absence of feedback from the recipient, i.e. by the non-arrival of ACKs. Note that the RTO timer is constantly updated during the lifetime of a TCP connection and its value at any moment in time is a function of the measured current RTT, hence its sole reliance on the end-to-end latency. The duration of time TCP waits for an ACK is known as the RTO timer value, which when it expires triggers the immediate retransmission of the first unacknowledged segment. Because the expiration of the RTO timer is typically an indication of severe congestion in traditional wired networks, TCP also drastically reduces its *cwnd* size, which usually results in a significantly reduced sending rate due to initiation of the slow start phase.

It has recently been derived that the traditional computation of TCP's RTO timer may not be suited to wired-to-wireless paths [145], which are typical of non-congestion related segment losses due to either radio channel corruptions or frequent

disconnections of devices from the AP. In addition, the 802.11 MAC at the AP uses persistent ARQ to recover from local frame errors, allowing a queue to develop in its send buffer for incoming higher-layer data. Each of these reasons causes randomly variable delays to occur, affecting TCP's end-to-end RTT estimate, which then affects the RTO timer calculation. Refer to Chapter 3 for a more details relating to the issues with the RTO timer over 802.11 WLANs.

To summarise then, observing the average RTO timer value of a particular TCP congestion control algorithm can be insightful, because its value is dependent on the current estimate of the RTT. Since each of the TCP algorithms on test utilises different methods for obtaining and updating its current RTT measurement, this could have a direct impact on values computed for the RTO timer. A large RTO timer value can help to absorb the high variability in RTTs due to the effects of the wireless path, by avoiding spurious timeouts and maintaining a higher overall sending rate. However, the RTO timer value should not be too large as to hinder TCP's fast response times in the recovery of segment losses that are genuinely occurring. There is clearly a trade-off between the increases in TCP throughput achieved due to avoiding spurious time-outs and the decreases in TCP throughput due to increased delays whilst waiting for an ACK to arrive and for the RTO timer to expire (i.e. the *cwnd* growth becomes stuttered as TCP is willing to wait a long time for an ACK to arrive).

7.4.1.2 Analysis of Effective FERs over the WLAN

Before proceeding with the actual experiments, the decision was made to conduct an assessment of channel error conditions over the WLAN for each of the sub-scenarios using the capabilities of the testbed. This would give insights into typical downlink and uplink FERs of a real-world home WLAN with a varying number of 802.11 end-devices.

For the experiments in this subsection, the iperf tool was used at the server to generate a continuous source of TCP data traffic (using a MSS of 1460 bytes) in a downlink fashion over the WLAN, with SEAMOSS as the receiving end-device. Each transfer lasted for 900 seconds, and was repeated five times for each sub-scenario. The AirPcap capture devices were configured to capture all forward and reverse data frames over

the WLAN, and all capture files were processed and analysed as per Chapter 6. All results are averages of the five runs for each sub-scenario.

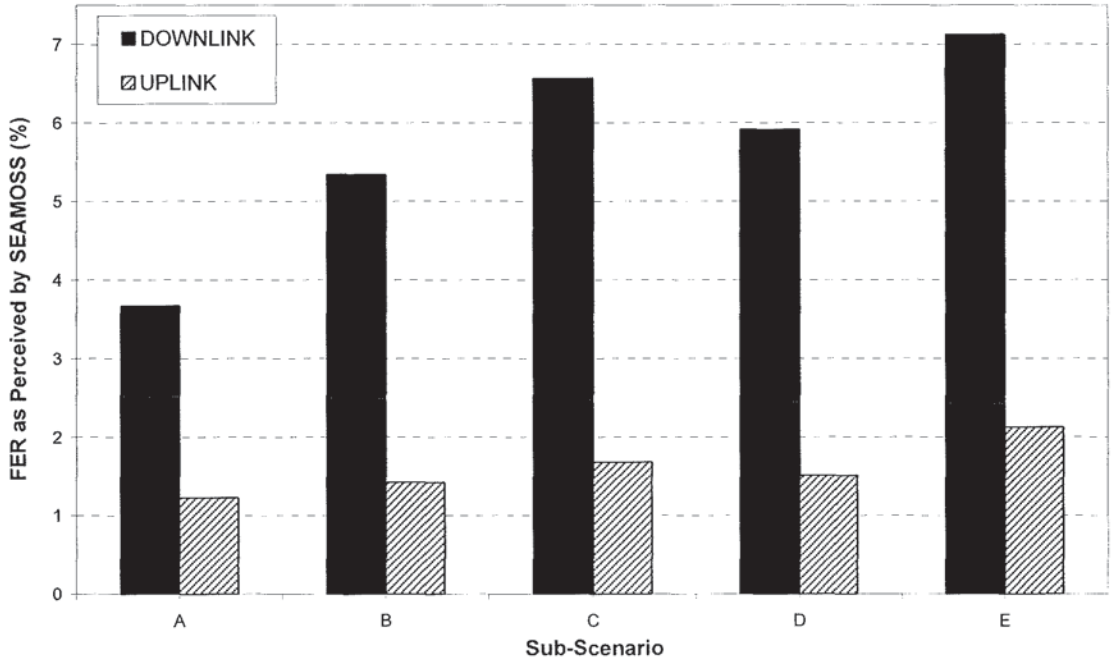


Figure 7.3: 802.11 FERs using TCP data flows under varying number of end-devices

The computed FERs in each sub-scenario for the downlink (containing TCP data segments) and uplink (containing TCP ACKs) channels of the WLAN have been plotted in Figure 7.3. Note that the WLAN was in saturation mode for all experiments. For each sub-scenario from A to E, the number of end-devices is incremented by one, up to a maximum of five. As can be seen from Figure 7.3, the FERs for the downlink channel are consistently higher than those for the uplink channel across all sub-scenarios. Interestingly, the downlink FERs possessed a more accelerated increase as the number of end-device increased, whereas the uplink FERs remained fairly steady up to sub-scenario D (four end-devices). In summary, from sub-scenarios A to E, the FERs for both channel directions almost doubled, with an error of over 7% affecting TCP data segments, and an error rate of over 2% affecting TCP ACKs. This can be explained by the fact that a greater number of end-devices generating TCP traffic in the WLAN creates greater levels of contention for access to the radio channel, and hence increases the likelihood of data frame collisions. The plot reveals that downlink TCP flows may suffer from greater unfairness than uplink TCP flows as the number of end-devices increases.

7.4.2 Scenario 2 – Varying the Location of the 802.11g End-Device

In this subsection there are three sub-scenarios that are considered; *good*, *fair*, and *poor* channel conditions. In the good sub-scenario the end-device SEAMOSS is located within the home so that its perceived SNR measurement is in the average region of 30 dB, as reported by the netstumbler tool [294]. In the fair sub-scenario SEAMOSS is located so that its experienced SNR measurement is in the region of 20 dB. In the poor sub-scenario SEAMOSS is located so that its perceived SNR measurement is in the region of 10 dB. Whilst sending TCP traffic to SEAMOSS, the TEMPLETREE and HANALENI devices were in their locations according to Table 7.2, performing the actions defined therein to generate additional WLAN traffic for added realism. Note that TEMPLETREE and HANALENI were both in saturation mode for all experiments in each sub-scenario.

7.4.2.1 Experiments and Selected Measurements

For each of three sub-scenarios, experiments consisted of transferring 30 Mb of TCP data using iperf from the experimental server to SEAMOSS in the last-hop WLAN. All data flows are in a downlink fashion. All TCP segments (including uplink TCP ACKs returning from SEAMOSS) passed through the Internet emulator machine. Each flow is therefore representative of an 802.11g end-user within a home environment downloading a 30 Mb file using TCP from a file-server located in the Internet. All flows were repeated three times for each TCP sender algorithm per sub-scenario. Hence all results are the averages of all measurements made.

Before proceeding with the actual experiments for the second main scenario, the decision was made to conduct an assessment of channel error conditions over the 802.11 WLAN for each of the channel condition sub-scenarios (good, fair, and poor conditions). This would give insights into typical downlink and uplink FERs for TCP traffic over a real-world home WLAN under varying channel conditions of a particular end-user.

As in the previous section, the iperf tool was used at the server to generate a continuous source of TCP data traffic (using a MSS of 1460 bytes) in a downlink

fashion over the WLAN, with SEAMOSS as the receiving end-device. Each transfer lasted for 900 seconds, and was repeated five times for each channel condition. The AirPcap capture devices were configured to capture all forward and reverse data frames over the WLAN, and all capture files were processed and analysed as per Chapter 6. All results are averages of the five runs for each sub-scenario.

7.4.2.2 Analysis of Effective FERs and Retransmission Characteristics

In Figure 7.4 the downlink and uplink FERs have been plotted for each of the perceived channel conditions by SEAMOSS in the home WLAN. The plot gives accurate insights into the FERs in a typical home WLAN scenario under varying conditions. As would be expected, when the SNR decreases the error rates will increase for both channel directions. An interesting observation is that the downlink FERs increased significantly for every 10 dB drop in the SNR, which wasn't uncommon between the different locations in the home shown in Figure 7.2. The uplink FERs also doubled for every 10 dB drop in the perceived SNR by SEAMOSS. Taking the poor channel conditions (i.e. two interior walls between SEAMOSS and the AP), the downlink FER was almost 65% and the uplink FER was almost 30%.

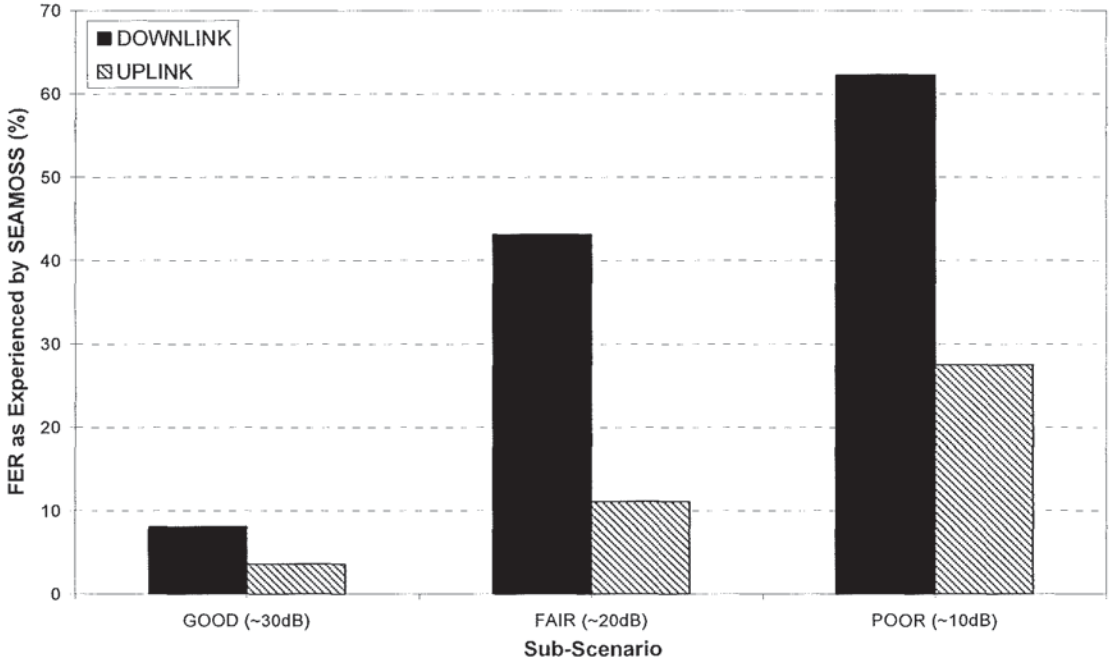


Figure 7.4: 802.11 FERs using TCP data flows over varying channel conditions

To gain further insights into the MAC retransmission characteristics behind the FERs presented in Figure 7.4, the distribution of retransmission probabilities have been calculated for both the AP and SEAMOSS over each sub-scenario. Figures 7.6 to 7.8 plot the probability distribution for all transmissions and retransmission attempts by the AP in the downlink channel (i.e. $P_{DOWNLINK}(i)$ where i is number of retransmission attempts made for a particular data frame). Figures 7.9 to 7.11 plot the probability distribution for all transmissions and retransmission attempts by SEAMOSS in the uplink channel (i.e. $P_{UPLINK}(i)$). To give further insights into the significance of i retransmission attempts of a data frame on the delays experienced by TCP traffic, all figures also plot the average transmission delay for an 802.11g sending MAC for i ranging from zero up to the retry limit. This delay value is, on average, the amount of time a TCP data segment or TCP ACK will require for a successful delivery across the radio channel, thereby impacting the RTT calculation by the TCP sender in the wired domain.

To calculate the theoretical delays experienced by each TCP data segment and TCP ACK, the 802.11 data frame exchange model needs to be referred to. Here it is assumed that each 802.11 data frame encapsulates only a single TCP data segment or TCP ACK. Recall that the 802.11 MAC requires a positive acknowledgement (an 802.11 ACK) for all data frames transmitted, and only proceeds to the next in-sequence frame after an 802.11 ACK has been received. If an 802.11 ACK is not received within the ACKTimeout interval, the MAC will retransmit the same data frame. This continues until the MAC reaches its retry limit.

For TCP transmissions over an 802.11 WLAN, the transactional model consists of data segments being sent in the downlink channel, and TCP ACKs being transmitted in the uplink channel. Therefore, according to the IEEE 802.11g standard [20], the delays associated with the transmission of a TCP data segment by the AP consists of the *Distributed Interframe Space* (DIFS) period, a random back-off period by the MAC, the transmission of the actual 802.11 data frame, a *Short Interframe Space* (SIFS) period, and the transmission of an 802.11 ACK by the end-device. Similarly for the 802.11 end-device, the transmission of a TCP ACK consists of delays associated with a DIFS period, a random back-off period by the MAC, the transmission of the actual 802.11 data frame, a SIFS period, and the transmission of an 802.11 ACK by the AP.

The aforementioned explanation is illustrated by means of a simplified diagram in Figure 7.5.

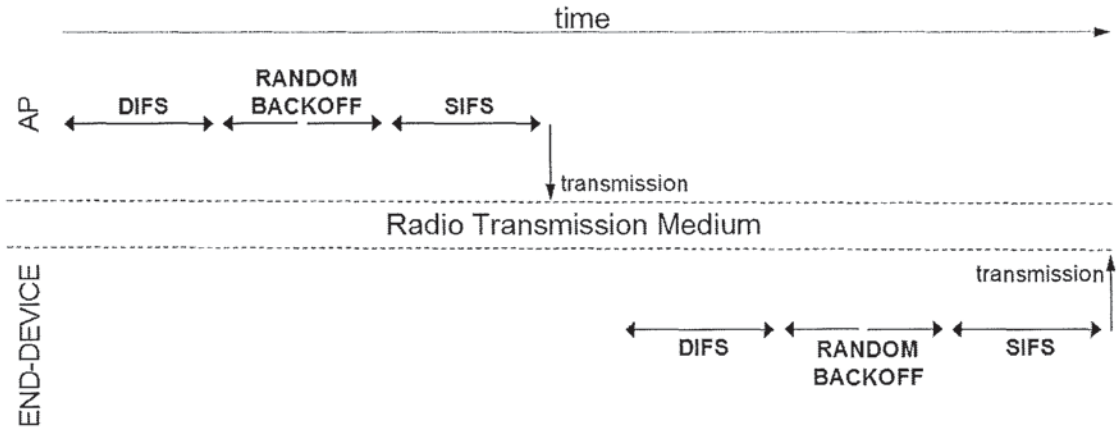


Figure 7.5: The 802.11 AP-client data interchange sequence

For TCP data segments of 1460 bytes, the total size of an 802.11 data frame transmitted in the downlink direction is 1536 bytes (12288 bits). For TCP ACKs of 40 bytes the total size of an 802.11 data frame transmitted in the uplink direction is 76 bytes (608 bits). Assuming that all 802.11 devices and the AP always transmit at a data rate of 54 Mbps, the times required to transmit a TCP data segment and a TCP ACK are given in Table 7.3 [23]. Table 7.3 also gives the duration values of the various 802.11g MAC transmission parameters, which have all been based on a data rate of 54 Mbps [121].

802.11g Parameter	Duration (ms)
802.11g_slotTime	0.009
802.11g_DIFS	0.028
802.11g_SIFS	0.010
Tx_802.11data_TCPdata	0.254
Tx_802.11data_TCPack	0.038
Tx_802.11ack	0.030
802.11_ACKtimeout	0.043

Table 7.3: 802.11g parameters and duration values

Prior to calculating the total transactional delays for downlink and uplink TCP traffic, it is important to understand and factor in additional delays associated with the random

and exponential back-off mechanism of the 802.11 MAC. Before gaining access to the radio medium for a transmission attempt, the sending MAC chooses to wait for a further back-off timer. The timer's value is set randomly by choosing an integer number of slots between zero and the minimum contention window (CW_{min}), which is multiplied by the slot-time defined for the 802.11g MAC. According to [297] the average number of slots chosen from the CW_{min}, μ, can be defined as:

$$\mu = \frac{1}{2} \cdot CW_{min} \qquad \textbf{(Eq. 7.1)}$$

where:

$$CW_{min} = \min(2^{(n+4)} - 1, 1023) \qquad \textbf{(Eq. 7.2)}$$

where n is the ordinal number of transmission attempts for a particular data frame. Hence, for the first transmission attempt, a sending 802.11g MAC's back-off timer will have an average value as defined by:

$$aBackoff = \mu \cdot slotTime \qquad \textbf{(Eq. 7.3)}$$

As soon as the back-off timer expires, the sending MAC is granted access to the medium and it transmits the frame. If an 802.11 ACK is not received within the ACKTimeout period, the MAC assumes that the frame transmission failed and schedules a retransmission by re-entering the back-off process described above. Table 7.4 presents the calculated values of the average back-off times in relation to the n^{th} frame transmission attempt for an 802.11g MAC using Eq. 7.3.

Assuming that the WLAN operates in an 802.11g-only mode (i.e. no 802.11b devices are present), and according to [297] the time required to transmit a TCP data segment over the WLAN is 0.254 ms, the time required to transmit a TCP ACK over the WLAN is 0.038 ms, and time required to transmit an 802.11 ACK is 0.03 ms.

Frame Transmission Attempt (n)	CW_{min}	Average Number of Slots (μ)	Average Back-off Time by MAC (ms)
1	15	7.5	0.06750
2	31	15.5	0.13950
3	63	31.5	0.28350
4	127	63.5	0.57150
5	255	127.5	1.14750
6	512	256	2.30400
7	1023	511.5	4.60350
> 7	1023	511.5	4.60350

Table 7.4: Calculated theoretical average back-off timer values for 802.11g MAC transmissions

Using the values listed in Table 7.4, the total average delay experienced by individual TCP data segments and TCP ACKs can be calculated. Taking the downlink channel initially, where TCP data segments are transmitted by the AP towards the end-device, the total MAC delay experienced in relation to the ordinal number of transmission attempts, n, is given by the following expressions (where the AP's retry limit is equal to 7):

When n is equal to 1 (i.e. the first transmission attempt) then:

$$APtotalMACdelay = DIFS + aBackoff + T_{TCPDATA} + SIFS + T_{802.11ACK} \quad \textbf{(Eq. 7.4)}$$

When n is equal to 2 (i.e. the second transmission attempt) then:

$$\begin{aligned} APtotalMACdelay &= DIFS + aBackoff(n=1) + T_{TCPDATA} \\ &+ ACKTimeout + DIFS + aBackoff(n=2) + T_{TCPDATA} + SIFS + T_{802.11ACK} \end{aligned} \quad \textbf{(Eq. 7.5)}$$

Continuing with the calculations given in Eq. 7.4 and Eq. 7.5, the average MAC delays at the AP for all values of n have been calculated and presented in Table 7.5, where the value of n is limited to 8 due to the retry limit. Looking at Table 7.5, it can be said that the average transmission delay experienced by a TCP data segment that requires, for

example, 6 transmission attempts of the encapsulating data frame is 5.95 ms (i.e. 5 retransmissions).

Total Transmissions (n)	Total Time Spent by AP MAC (ms)
1	0.39
2	0.85
3	1.46
4	2.36
5	3.58
6	5.95
7	10.63
8	15.30

Table 7.5: Theoretical (downlink) transmission delays for TCP data segments at the AP

Taking the uplink TCP ACK transmissions towards the AP, the total MAC delay experienced in relation to the n^{th} transmission attempt is given by the following expressions (where the end-device's retry limit is equal to 15):

When n is equal to 1 (i.e. the first transmission attempt) then:

$$EndDevicetotalMACdelay = DIFS + aBackoff + T_{TCP\,ACK} + SIFS + T_{802.11\,ACK} \qquad \textbf{(Eq. 7.6)}$$

When n is equal to 2 (i.e. the second transmission attempt) then:

$$\begin{aligned} EndDevicetotalMACdelay &= DIFS + aBackoff(n=1) + T_{TCP\,ACK} \\ &+ ACKTimeout + DIFS + aBackoff(n=2) + T_{TCP\,ACK} + SIFS + T_{802.11\,ACK} \end{aligned} \qquad \textbf{(Eq. 7.7)}$$

Continuing with the calculations given in Eq. 7.6 and Eq. 7.7, the average MAC delays at the end-device for all values of n have been calculated and presented in Table 7.6, where the value of n is limited to 16 due to the retry limit. Looking at Table 7.6, it can be said that the average transmission delay experienced by a TCP ACK that requires, for example, 11 transmission attempts of the encapsulating data frame takes 28.46 ms (i.e. 10 retransmissions).

Total Transmissions (n)	Total Time Spent by End-Device MAC (ms)
1	0.17
2	0.42
3	0.81
4	1.50
5	2.71
6	5.09
7	9.76
8	14.44
9	19.11
10	23.79
11	28.46
12	33.14
13	37.81
14	42.48
15	47.16
16	51.83

Table 7.6: Theoretical (uplink) transmission delays for TCP ACKs at the end-device

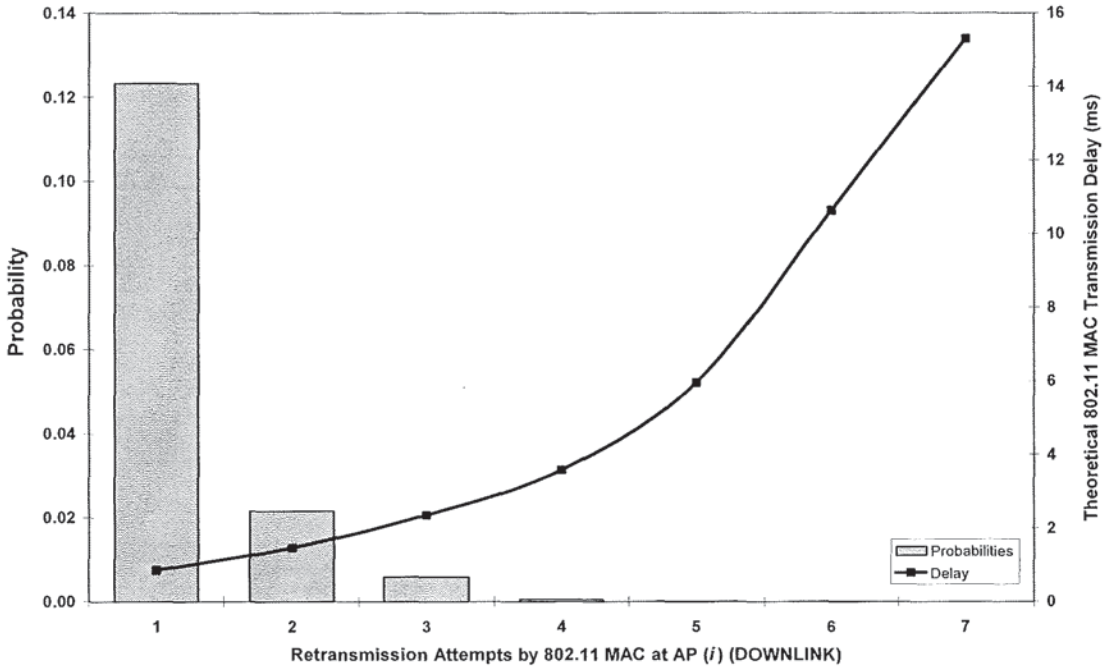


Figure 7.6: Retransmission probabilities of the AP in good conditions (SNR~30dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

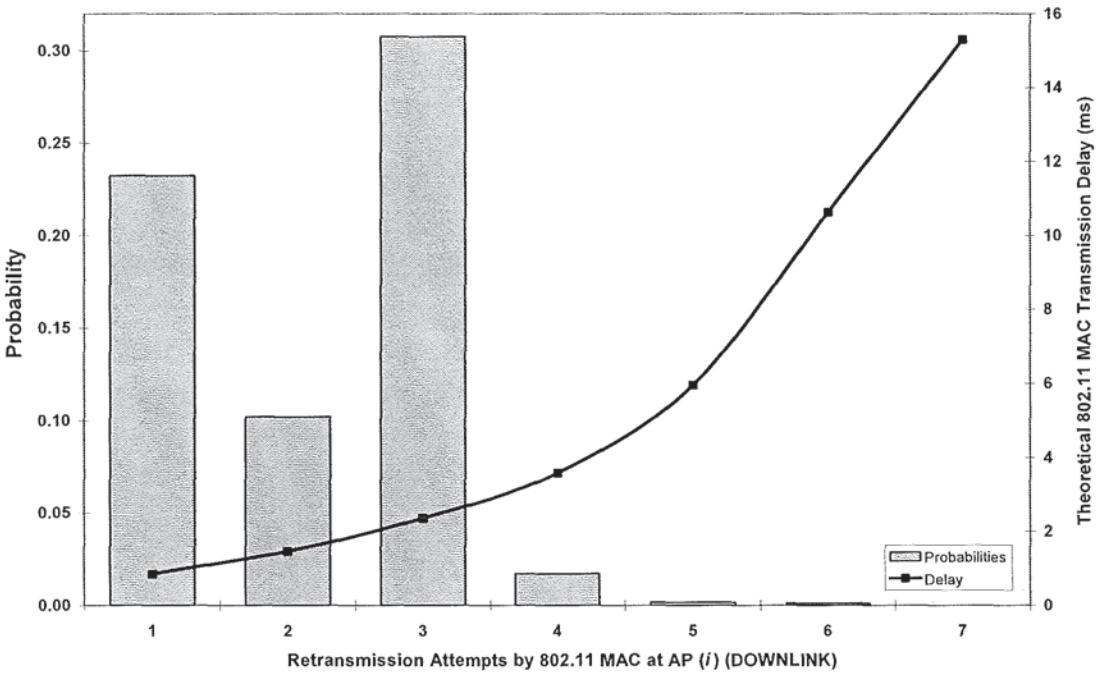


Figure 7.7: Retransmission probabilities of the AP in fair conditions (SNR~20dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

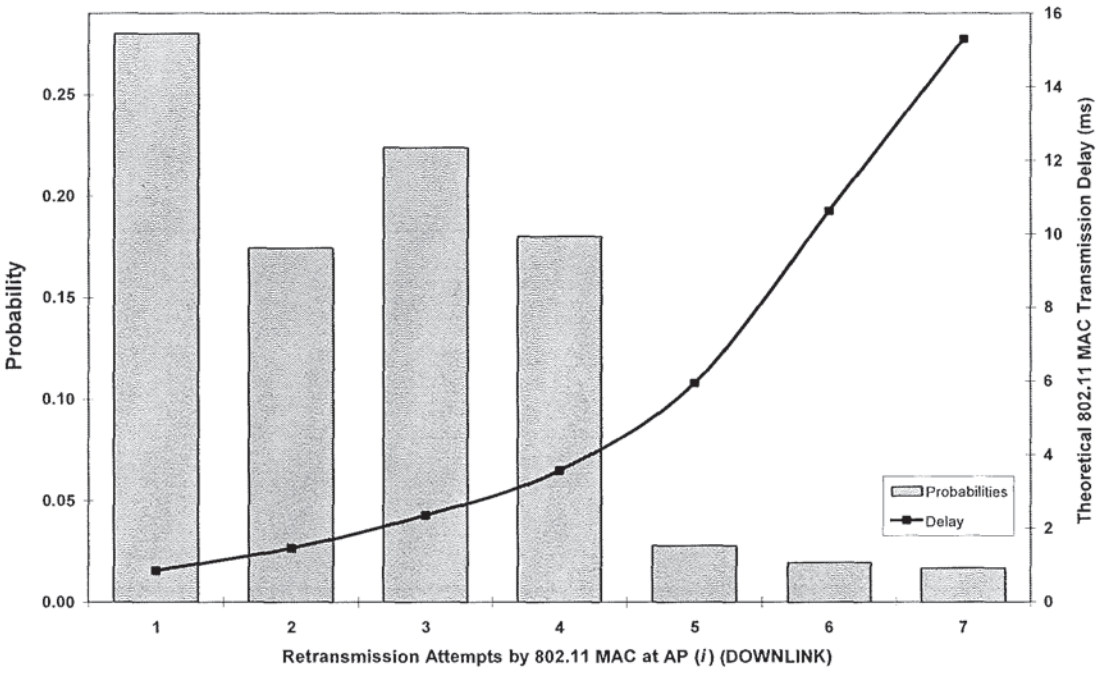


Figure 7.8: Retransmission probabilities of the AP in poor conditions (SNR~10dB)

(Probabilities plotted using $P_{DOWNLINK}(i) \times FER_{DOWNLINK}$)

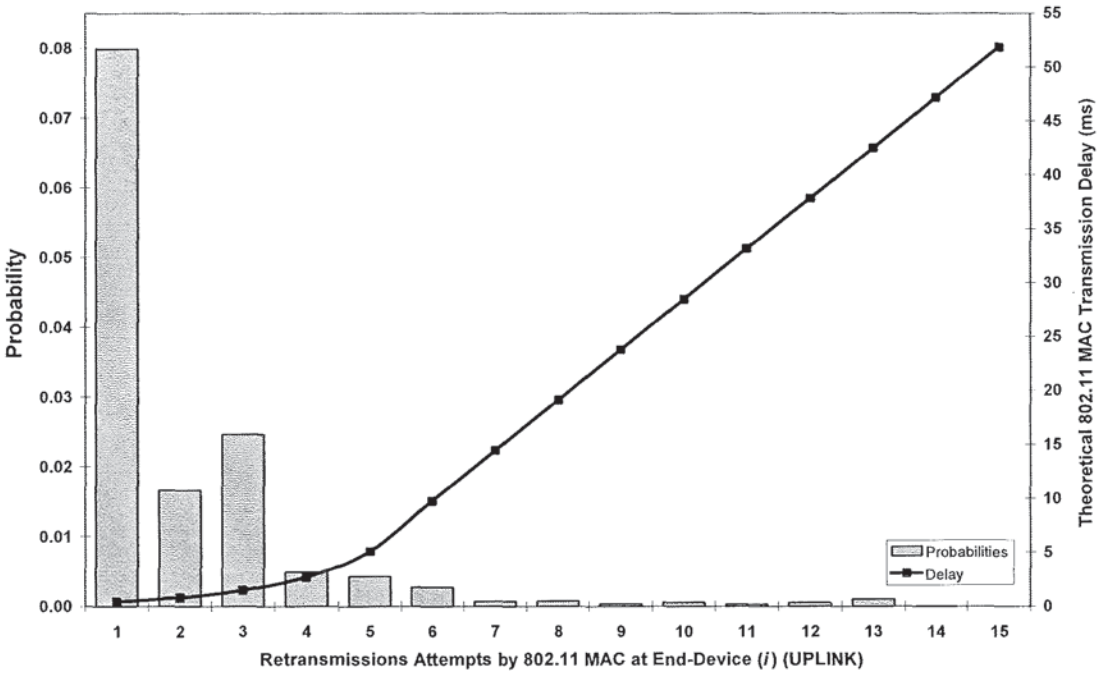


Figure 7.9: Retransmission probabilities of SEAMOSS in good conditions (SNR~30dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

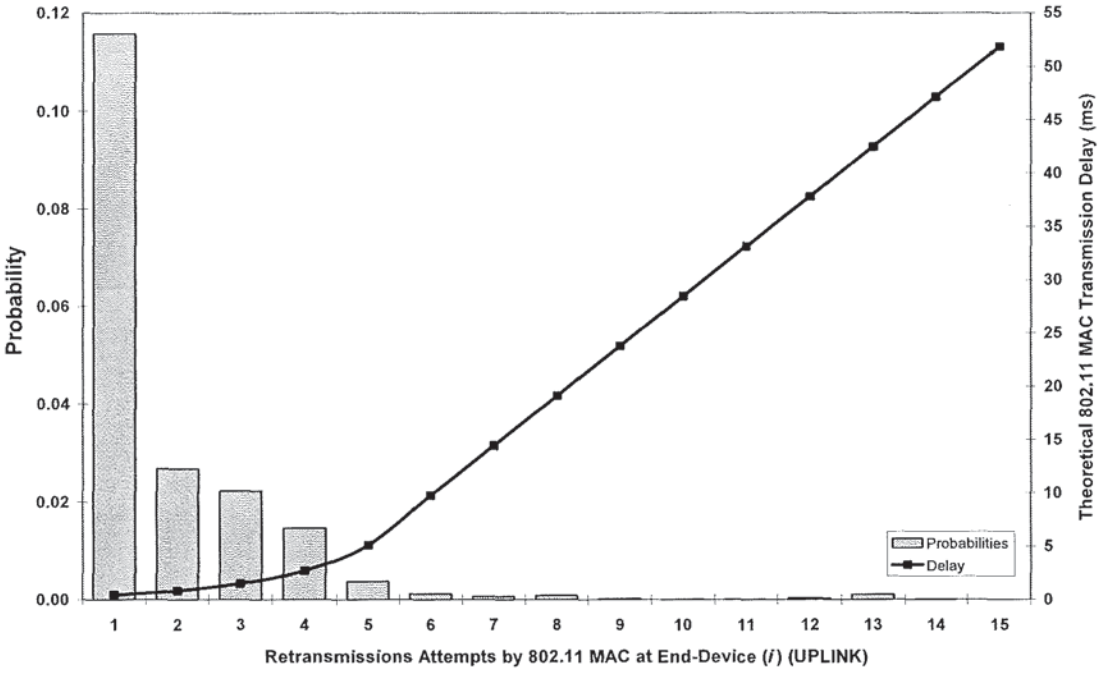


Figure 7.10: Retransmission probabilities of SEAMOSS in fair conditions (SNR~20dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

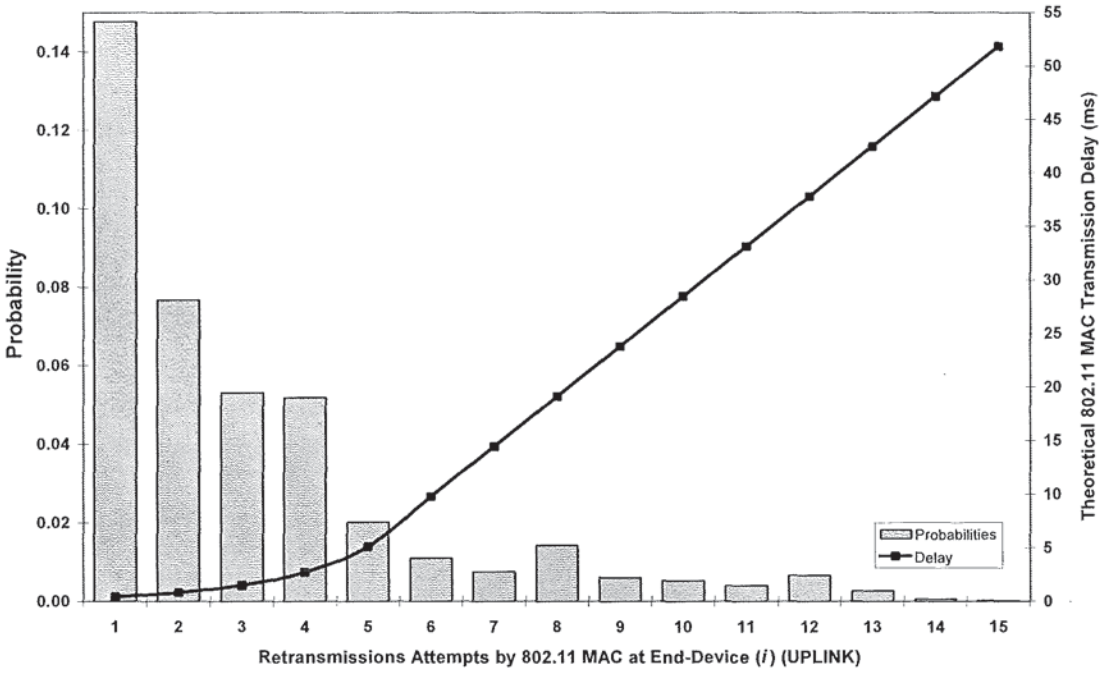


Figure 7.11: Retransmission probabilities of SEAMOSS in poor conditions (SNR~10dB)

(Probabilities plotted using $P_{UPLINK}(i) \times FER_{UPLINK}$)

In Figure 7.6 for the downlink retransmission probabilities in good channel conditions, it can be seen that 10% of all unique data frames (TCP data segments) required one retransmission before success by the AP. It can also be said that 10% of all unique TCP data segments arriving at the AP from the server experienced an average delay of 0.85 ms over the downlink channel, and 2% of all unique TCP data segments experienced an average delay of 1.46 ms over the downlink channel (due to them requiring two retransmission attempts by the AP).

Looking at Figure 7.7 for the downlink retransmission probabilities in fair channel conditions, the results are very different. Over 20% of unique data frames required one retransmission by the AP, 10% required two retransmissions, and over 30% required three retransmission attempts before successfully arriving at SEAMOSS. In other words, 30% of all unique TCP data segments arriving at the AP experienced an average transmission delay of 2.36 ms before arriving at SEAMOSS.

Looking at Figure 7.8 for the downlink retransmission probabilities in poor channel conditions, the results are more pronounced, with i equalling 5, 6, and 7 all having probabilities in the region of 2% to 3%. Hence, around 2% of all unique TCP data

segments arriving at the AP experienced an average transmission delay over the forward channel of 10.63 ms, with a further 2% experiencing an average transmission delay of 15.3 ms.

Figures 7.9 to 7.11 give insights into the distribution of retransmission probabilities (and associated TCP ACK transmission delays) for SEAMOSS, who was transmitting TCP ACKs back to the AP in an uplink fashion. In the good channel conditions (Figure 7.9) the distribution of probabilities for i in the uplink are comparable to that for the downlink. Under fair channel conditions (Figure 7.10), the distribution of probabilities for i are little affected, remaining similar to those seen under good channel conditions. In Figure 7.11 for poor channel conditions, it can been seen that over 5% of all unique TCP ACKs experienced transmission delays at the MAC of SEAMOSS of 1.46 ms, with around 5% experiencing delays of 2.36 ms, and a further 5% experiencing delays of 3.58 ms. It should also be noticed that due to the higher default retry limit (15) of the 802.11g adapter in SEAMOSS, 2% of all unique TCP ACKs experienced delays of 19.11 ms (i.e. 8 retransmission attempts of the same data frame), and 1% experienced delays of 37.81 ms before successfully reaching the AP (i.e. 12 retransmission attempts of the same data frame).

7.4.2.3 Analysis of Transmission Delays for TCP Traffic over the WLAN

To summarise the findings from the preceding subsection on delays experienced by downlink and uplink TCP traffic, Figure 7.12 plots the average transmission delays experienced by unique TCP data segments over the downlink channel and for TCP ACKs in the uplink channel. The averages have been calculated by summing up all of the individual TCP delay samples reported from the experimental testbed's software tools, and then dividing by the number of samples reported. Hence, Figure 7.12 presents an accurate view from the analysis of raw data values extracted from the captured 802.11 data frames in both channel directions. Looking at the plot, it can be said that overall TCP ACKs in the uplink channel experienced around twice the delay as TCP data segments in the downlink channel. This could be explained by the higher retry limit of SEAMOSS, making its ARQ mechanism more persistent than it is necessary in good and fair channel conditions. It can be seen that in good and fair conditions, the average delays experienced by TCP traffic is no more than 200 ms in

each direction. However, in poor channel conditions, downlink TCP data segments experienced an average transmission delay at the AP of almost 2.1 seconds, and very surprisingly TCP ACKs in the uplink direction experienced average delays of almost 8 seconds before successfully reaching the AP.

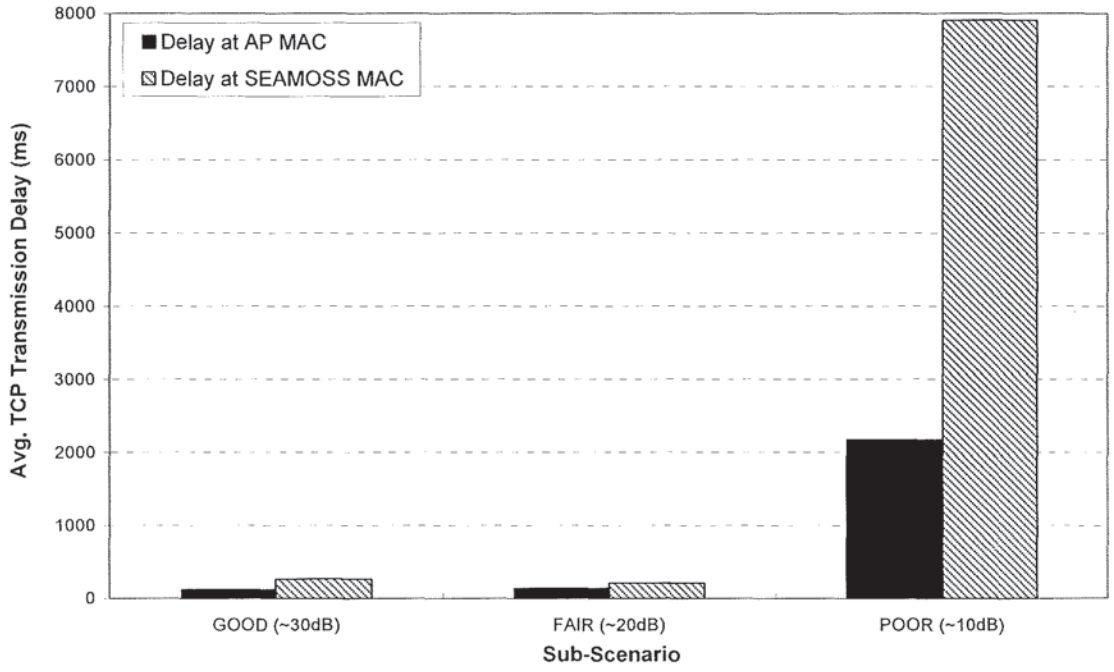


Figure 7.12: Average transmission delays for downlink and uplink TCP traffic

These values can be quite severe for a TCP sender in the Internet as on average it will be waiting for over 10 seconds from the point of sending a TCP data segment until the associated ACK arrives. Of course, the sender is more likely to experience an expiry of its RTO timer under such conditions, leading to an unnecessary retransmission and *cwnd* reduction potentially.

Using the individual transmission delay samples reported from the testbed for downlink and uplink TCP traffic over the WLAN, further analysis work was carried out that sorted through all the samples and generated the individual probabilities associated with a particular delay value occurring at the MAC of the AP and SEAMOSS. Figures 7.13, 7.15, and 7.17 plot the delay probabilities for downlink TCP data segment transmissions by the AP under good, fair, and poor channel conditions, respectively. Figures 7.14, 7.16, and 7.18 plot the delay probabilities for uplink TCP ACK transmissions by SEAMOSS under good, fair, and poor channel conditions, respectively. Such accurate insights into the actual probabilities of a certain amount of

delay occurring could assist researchers with fine tuning TCP's RTO timer to prevent unnecessary timeout events by referencing the likelihood cause of a sudden delay.

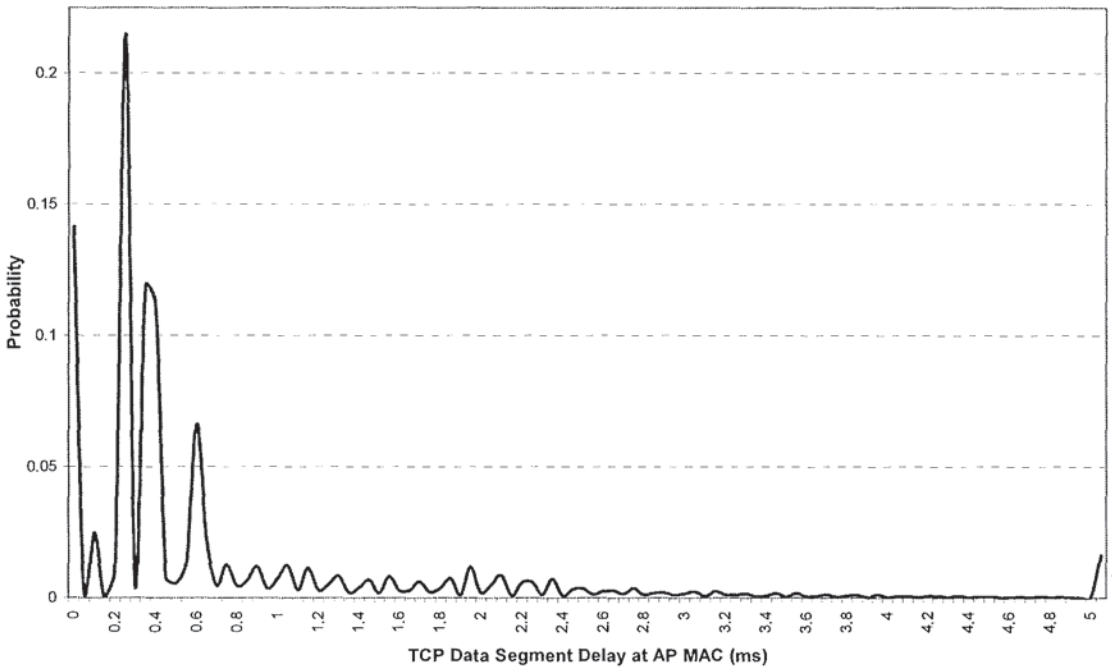


Figure 7.13: Probabilities of transmission delays at AP in good conditions (SNR~30dB)

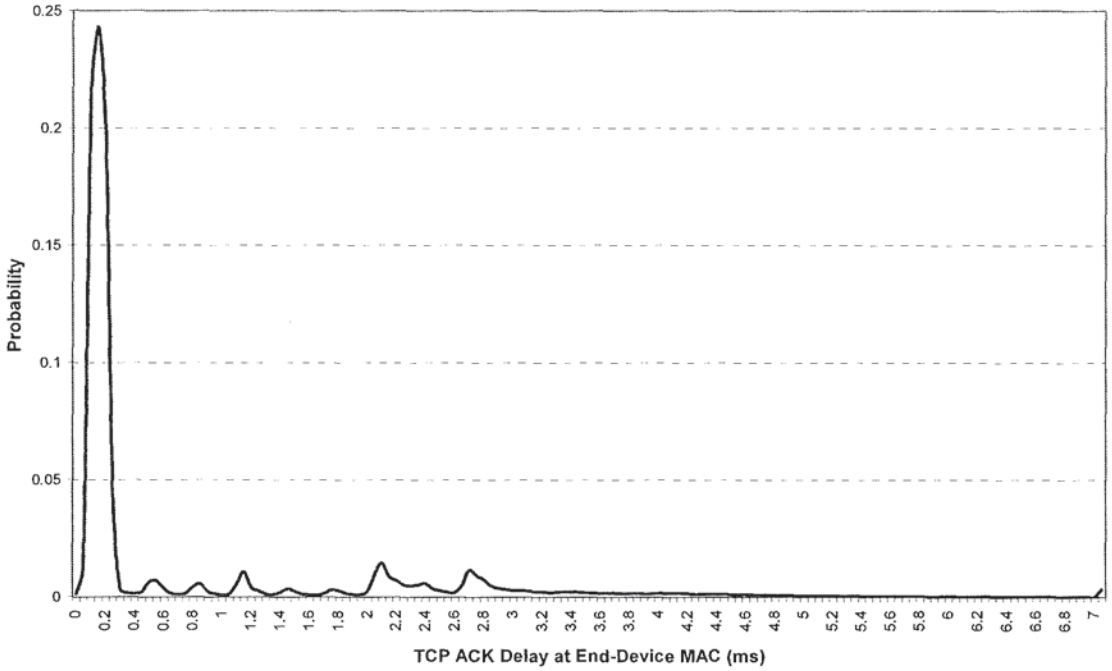


Figure 7.14: Probabilities of transmission delays at SEAMOSS in good conditions (SNR~30dB)

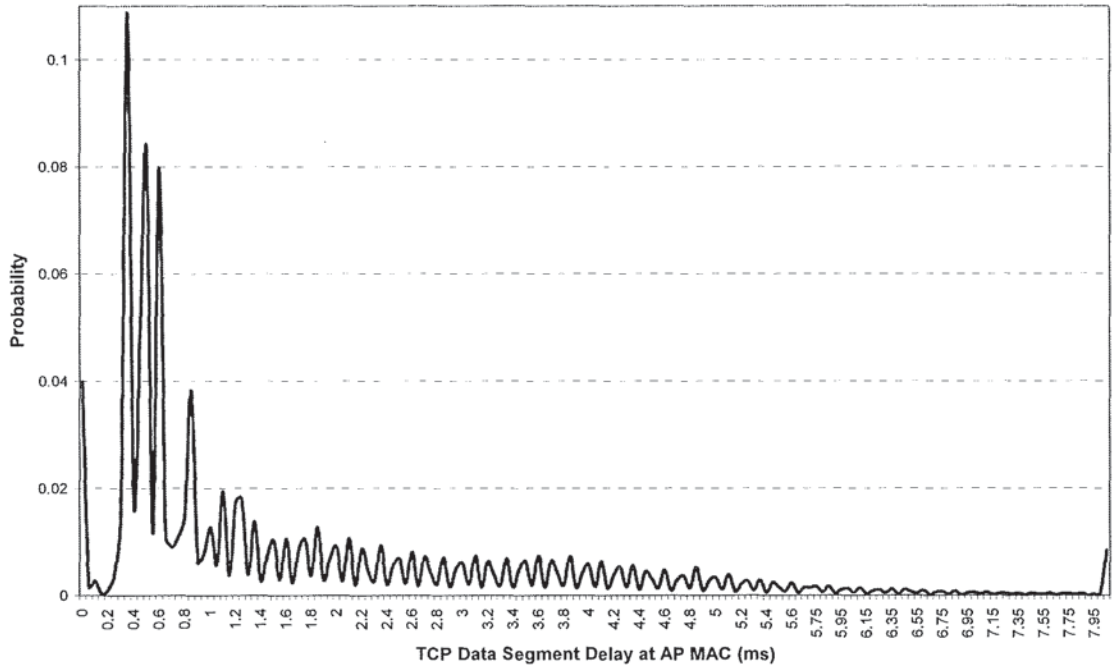


Figure 7.15: Probabilities of transmission delays at AP in fair conditions (SNR~20dB)

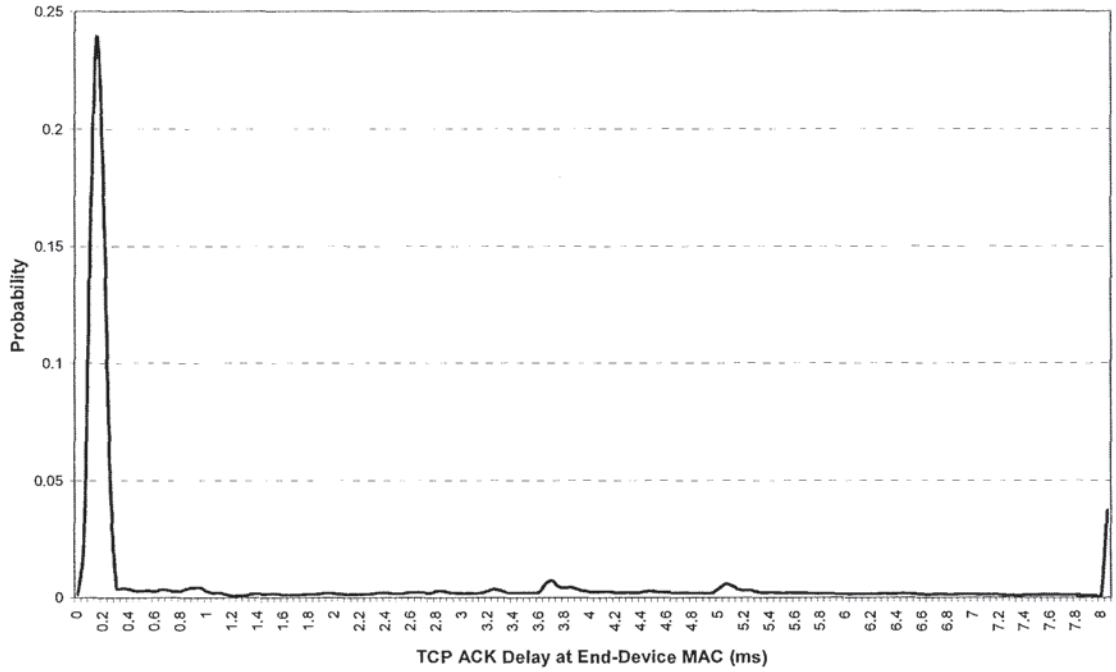


Figure 7.16: Probabilities of transmission delays at SEAMOSS in fair conditions (SNR~20dB)

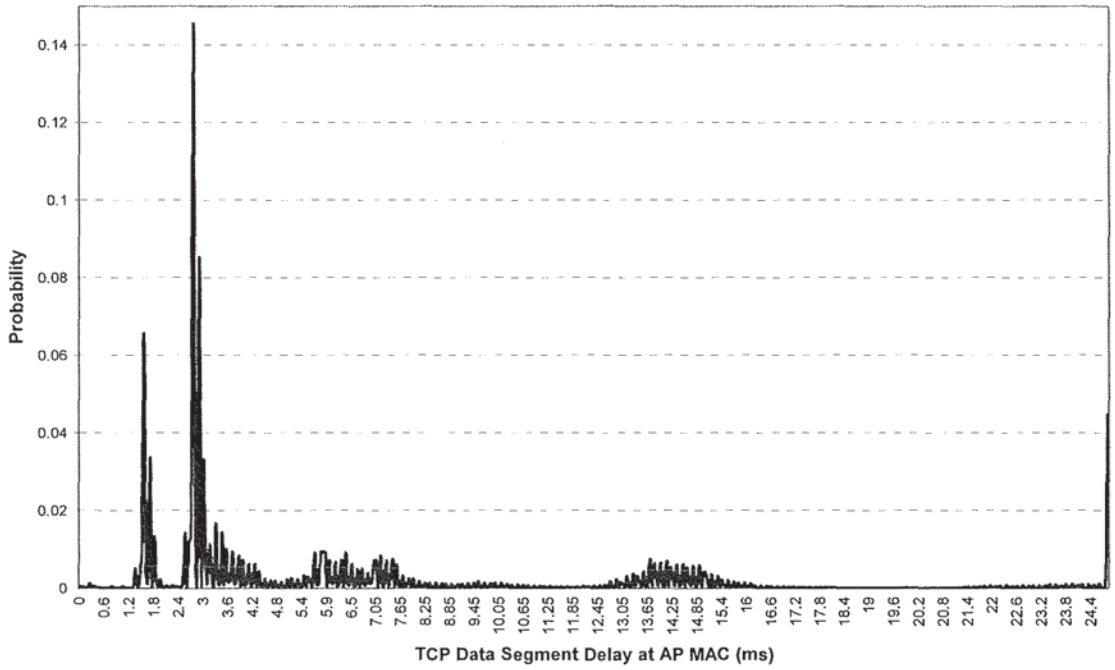


Figure 7.17: Probabilities of transmission delays at AP in poor conditions (SNR~10dB)

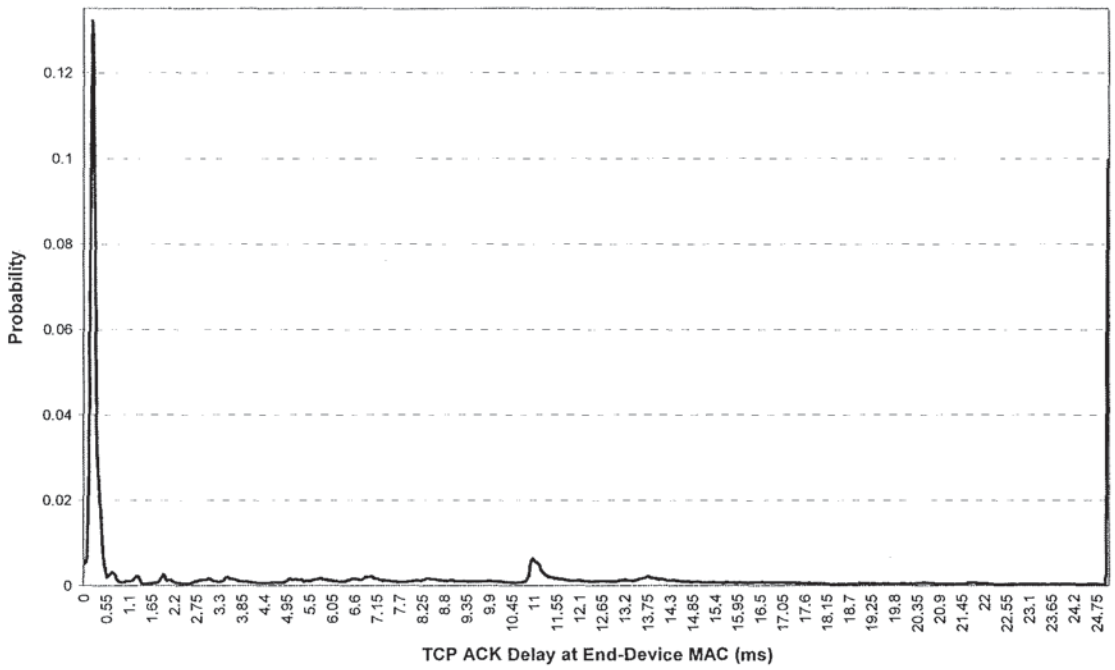


Figure 7.18: Probabilities of transmission delays at SEAMOSS in poor conditions (SNR~10dB)

From Figures 7.13 to 7.18, it is interesting to observe the changing magnitude of the delay values along the horizontal axes of the plots. This happens at the interchange between good conditions and fair conditions, and between fair conditions and poor conditions. A TCP sender could therefore be enhanced with heuristics that allow it to

estimate whether conditions in the last-hop WLAN are in good, fair, or poor state. Based on which state is estimated, a sending TCP could use statistics presented in this subsection to take different approaches for determining its RTO timer, and/or react differently to segment losses and non-arriving ACKs within a specific time period.

7.5 Scenario 1 – Results and Discussions

In this section the results obtained from performing end-to-end TCP data transfers to SEAMOSS in the last-hop WLAN are presented. The effectiveness of TCP Reno, TCP CUBIC, TCP Hybla, TCP Veno, and TCP Westwood+ over the wired-to-wireless testbed, and for differing sub-scenarios over the WLAN, is also reported.

7.5.1 Small TCP Transfers

Here the results for the 1 Mb data transfers between the experimental TCP server and SEAMOSS are presented. The results incorporate data transfers for sub-scenario A (consisting of SEAMOSS only) to sub-scenario E (consisting of five 802.11g devices in the WLAN including SEAMOSS). The averages of the results from three independent iperf runs per TCP congestion control variant per sub-scenario are presented.

Figure 7.19 is a plot of the average time taken to transfer 1 Mb of data between the server and SEAMOSS, as reported by the iperf tool running on SEAMOSS. It can be seen that Hybla supersedes all of the other TCP congestion control algorithms on test for all sub-scenarios. The legacy Reno also performed the transfer in less time than Veno, TCP Westwood+, and CUBIC in sub-scenarios A, B, and C. This suggests that for short transfers and fewer 802.11g devices in the WLAN, Reno's performance may not suffer as prominently as might originally have been thought. However, as the number of devices in the WLAN was increased beyond three, Reno's performance quickly degraded the most, as can be seen from the rapid increase in the transfer time. Interestingly, throughout the tests Veno performed only slightly better than Reno; this is to be expected because it is a direct modification of the AIMD algorithms used in Reno, with even identical functionality under certain circumstances.

Figure 7.20 plots the average end-user throughput achieved during the lifetime of a particular transfer for each of the sender-side algorithms on test. It was calculated by using tcptrace on the tcpdump capture files obtained from SEAMOSS. The tcptrace tool reported the total number of TCP data segments that were actually received at the TCP-layer, denoted by N_{TCP}. The time taken to transfer the entire 1 Mb of data according to the iperf tool is denoted by T. The TCP throughput (bps), η, was obtained using the following equation (where MSS = 1460× 8 = 11,680 bits):

$$\eta = \frac{N_{TCP} \cdot MSS}{T} \qquad \textbf{(Eq. 7.8)}$$

Note that Eq. 7.8 is a more accurate computation of the perceived TCP throughput for the end-users in the WLAN, as it excludes the sizes of all lower-layer packet headers as well as the actual TCP header. It focuses solely on the payload, which is the only data that is passed to the application-layer.

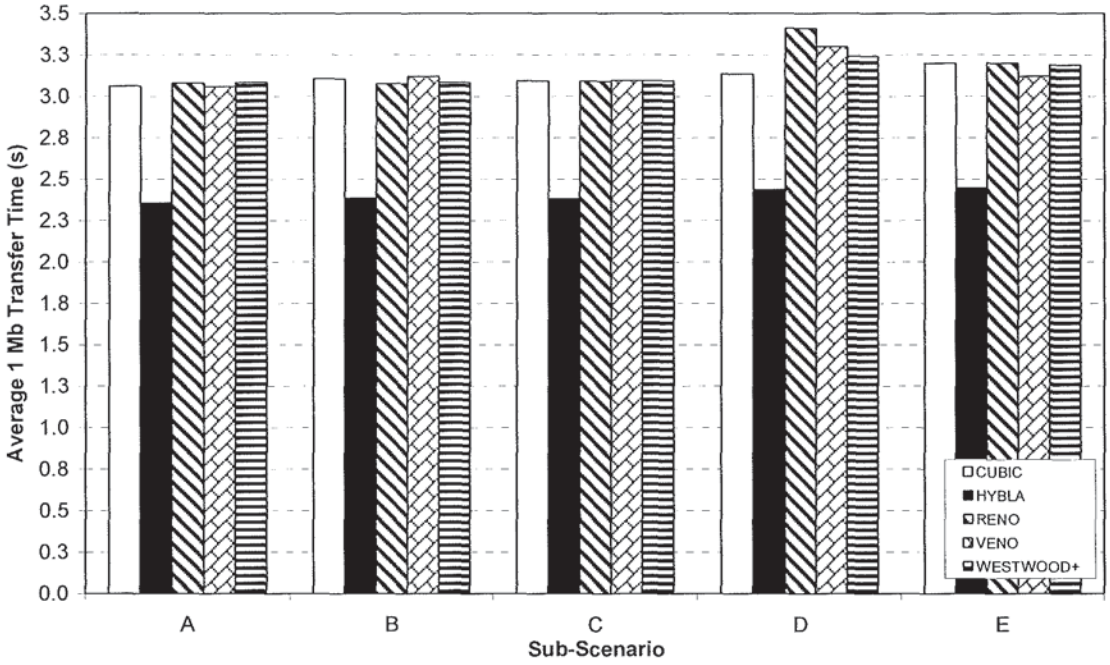


Figure 7.19: Short TCP flows: Average 802.11g end-user download time for 1 Mb of data

It can be seen from Figure 7.20 that all TCP congestion control algorithms experienced a drop in average throughput as the number of devices in the last-hop WLAN increased, from sub-scenario A to D. However, an interesting phenomenon can be seen

with TCP Reno, Veno, and Westwood+ in that they all experienced a slight increase in their average throughput in sub-scenario E, i.e. when the number of 802.11g devices was increased from four to five. TCP CUBIC and Hybla however continued to show a decrease, as would be expected due to the higher end-to-end latencies being caused by increasing contention levels over the WLAN between the AP and 802.11g devices for access to the radio medium [53]. Since each of the TCP congestion control algorithms on test relies on its estimate of the RTT for advancing its *cwnd* size, one would expect a gradual decrease in end-to-end throughput between the TCP server and SEAMOSS as the number of 802.11g devices increase, as the *cwnd* is growing at a reduced rate. Another factor that may have affected the average end-to-end throughput is that of the increased data frame errors over the WLAN due to an increase in the number of 802.11g devices and the subsequent increase in the amount of TCP traffic being generated over the downlink and uplink channels (as is shown in Figure 7.3). According to [41] and [258], 802.11 frame errors tend to be more prevalent as the number of 802.11g devices and hence TCP traffic levels increase; this is due to the effects of 'self collisions' of 802.11 data frames containing TCP segments.

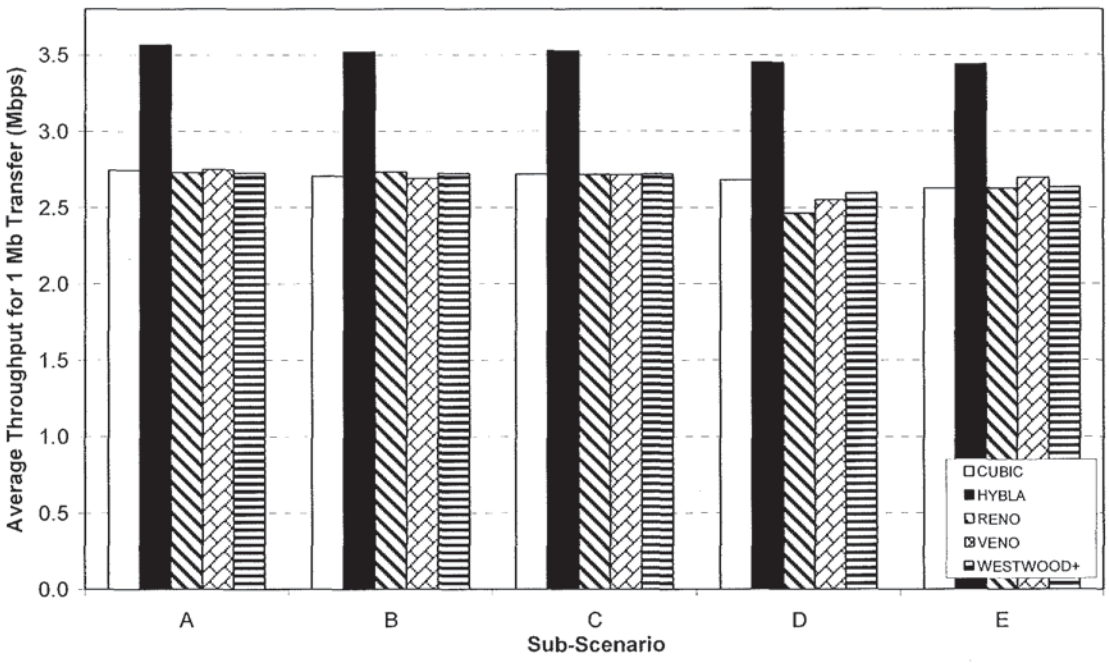


Figure 7.20: Short TCP flows: Average throughput achieved by 802.11g end-user

The primary explanation here is that TCP ACKs from 802.11g devices are sent as 802.11 data frames in the uplink direction of the WLAN. With 802.11 data frames

travelling in both uplink and downlink directions, and with multiple devices at various locations in the home WLAN, frame collisions can occur due to the 'hidden node problem' and frames are corrupted (refer to Chapter 3). Of course, the 802.11 sending MAC is able to conceal the majority of frame errors using its stop-and-wait ARQ mechanism, however not all data frames can be recovered because the MAC will only attempt a fixed number of retransmissions before giving up and moving onto the next in-sequence data frame in its transmit queue. This effectively translates into a lost TCP data segment that will not arrive at SEAMOSS, or a lost TCP ACK that will not arrive at the AP. A lost TCP data segment will cause SEAMOSS to generate DUPACKs for every out-of-order data segment arriving thereafter, which will force the TCP sender to perform a fast retransmit and fast recovery action, reducing its *cwnd* size, and consequently its sending rate, unnecessarily. A lost TCP ACK will cause the TCP sender's RTO timer to expire as it waits for it to arrive, which incorrectly retransmits the presumed lost data segment, and again reduces its *cwnd* size, and its sending rate.

The most striking result in Figure 7.20 is that TCP Hybla achieves the highest average throughput across all sub-scenarios, producing up to a 30% better performance than the next best algorithm, CUBIC. In sub-scenario E however, Veno achieves a higher throughput than CUBIC, perhaps suggesting its superiority when the number of 802.11g devices increases significantly, which would require further investigation for confirmation.

7.5.2 Medium TCP Transfers

Here the results for 10 Mb transfers between the TCP server and SEAMOSS in the last-hop WLAN are provided, again presenting averages of three independent runs per TCP congestion control variant, and per sub-scenario.

Figure 7.21 is a plot of the average times taken to transfer 10 Mb of data for each of the TCP algorithms on test. It can be seen that all TCP algorithms, except for Reno, follow a similar trend between sub-scenarios A to E, possessing a gradual increase in the time taken to complete the transfer as the number 802.11g devices in the WLAN is increased. Hybla performed the best, by completing the transfer, on average, 1 second quicker in all sub-scenarios. In contrast, Reno exhibited a significant increase in its

transfer time from sub-scenario B to D inclusive; in sub-scenario C it took nearly 2 seconds (~7%) longer than Hybla to transfer the 10 Mb of data; and in sub-scenario D it took nearly 5 seconds (~22%) longer. Interestingly however, Reno performed better than Westwood+ and CUBIC in sub-scenario E, where there were five 802.11g devices in the WLAN. A possible explanation for this sudden increase in performance could be due to the way in which the 802.11g MAC randomly allocates back-off periods from its contention window before gaining access to the radio medium [23]. Thus, five devices in the WLAN may have altered the MAC's randomness in favour of SEAMOSS acquiring access to the radio medium quicker than the other devices. Another possible explanation could be that Reno's calculation of the RTO timer is more sensitive to conditions in the last-hop WLAN, such as the loss rates affecting TCP traffic and the variability of delays over the WLAN.

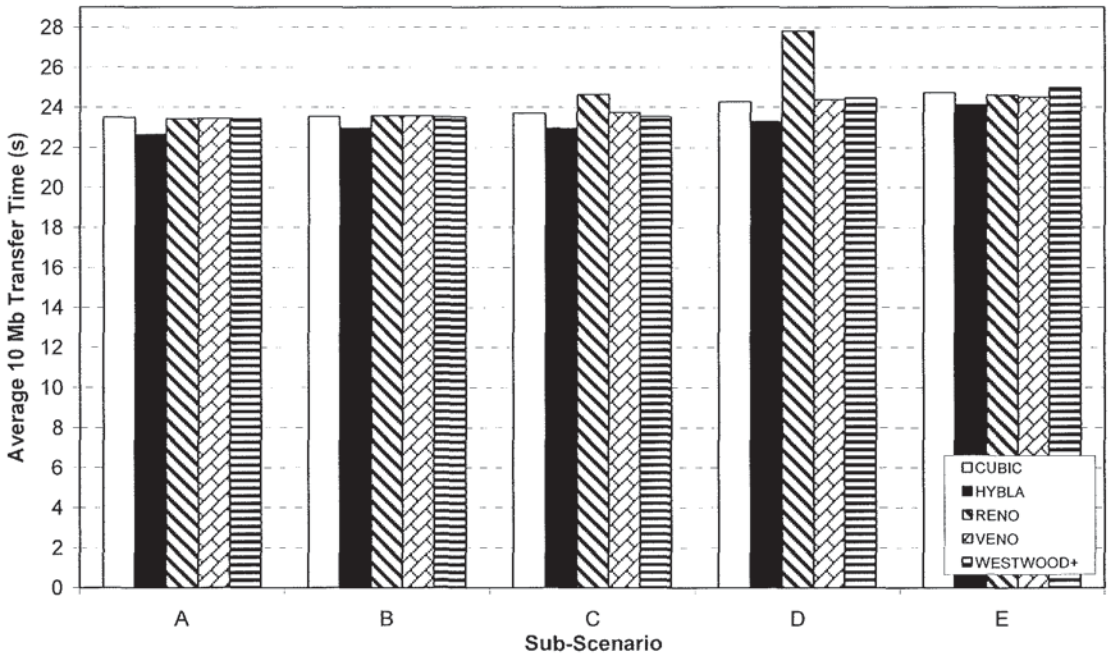


Figure 7.21: Medium TCP flows: Average 802.11g end-user download time for 10 Mb of data

Figure 7.22 plots the average throughput achieved by the end-user over the lifetime of a 10 Mb data transfer to SEAMOSS. Again, as elaborated upon in the preceding subsection, tcptrace was used in conjunction with tcpdump capture files from SEAMOSS. As already discussed for Figure 7.21, TCP Hybla achieved the highest throughput across all sub-scenarios. Because the congestion avoidance phase was more prevalent in these transfers, all TCP algorithms were able to probe for higher

bandwidth in all sub-scenarios due to way in which the AIMD mechanism has been designed; hence the throughput values in Figure 7.22 are greater than those seen in Figure 7.20. It was also noticed that up to sub-scenario C (i.e. three 802.11g devices in the WLAN), all TCP algorithms experienced only a mild decrease in average end-user throughput. When introducing a fourth (sub-scenario D) and fifth (sub-scenario E) device into the WLAN, the decrease in throughput was much more accelerated, with some more accentuated than others. Hence, it could be said from these results that medium size TCP transfers are little affected by last-hop WLANs containing up to three 802.11g devices, but show a more rapid decline in performance when the number of devices exceeds this number.

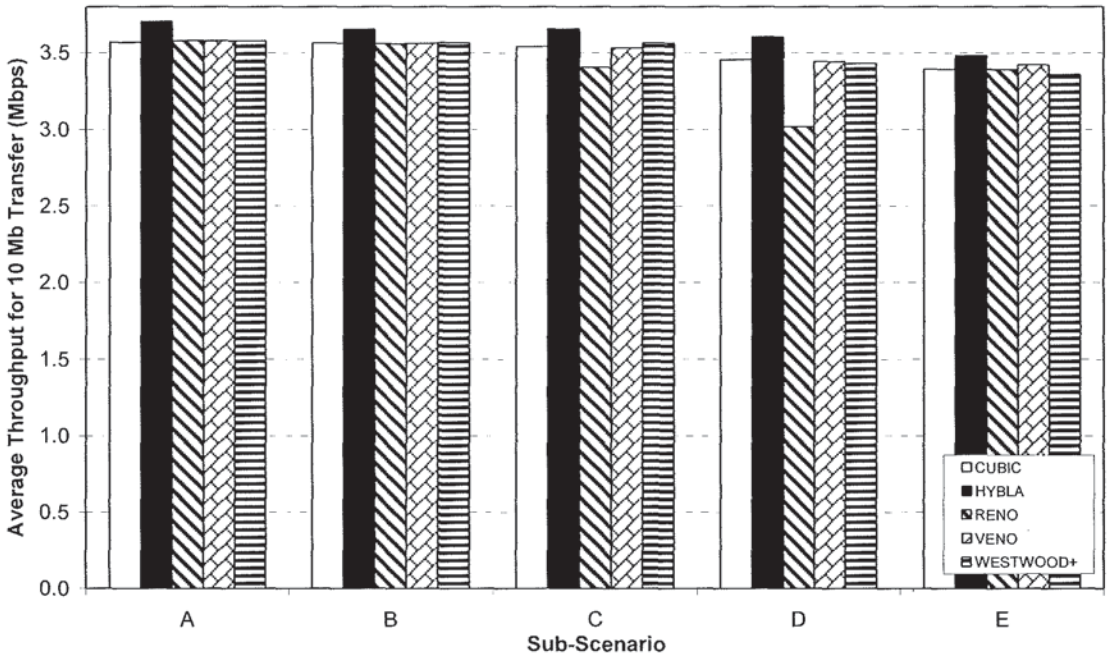


Figure 7.22: Medium TCP flows: Average throughput achieved by 802.11g end-user

7.5.3 Large TCP Transfers

Here the results for 50 Mb data transfers between the TCP server and SEAMOSS in the last-hop WLAN are provided, again presenting averages of three independent runs per TCP congestion control variant, and per sub-scenario.

Figure 7.23 is a plot of the average times taken to transfer 50 Mb of data for each of the TCP algorithms on test, and Figure 7.24 is a plot of the average end-user

throughput achieved over the lifetime of connections. It can be seen that from sub-scenarios A to C, the average throughput (and subsequently transfer times) for each algorithm remains relatively constant, around the 3.7 Mbps to 3.75 Mbps (around the 114 seconds) mark.

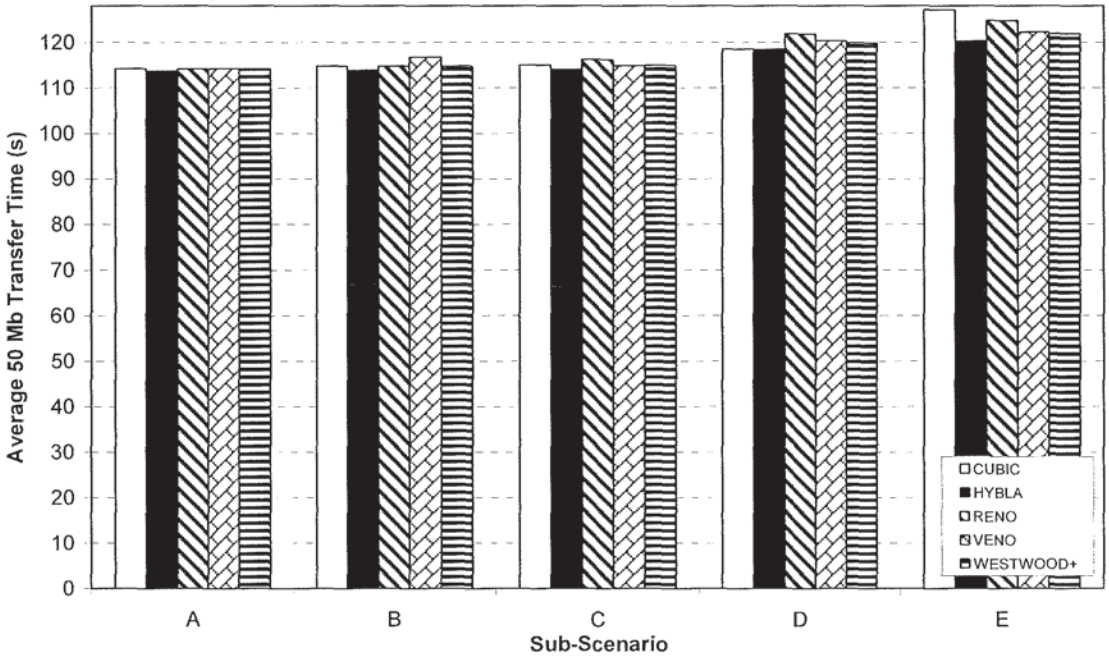


Figure 7.23: Large TCP flows: Average 802.11g end-user download time for 50 Mb of data

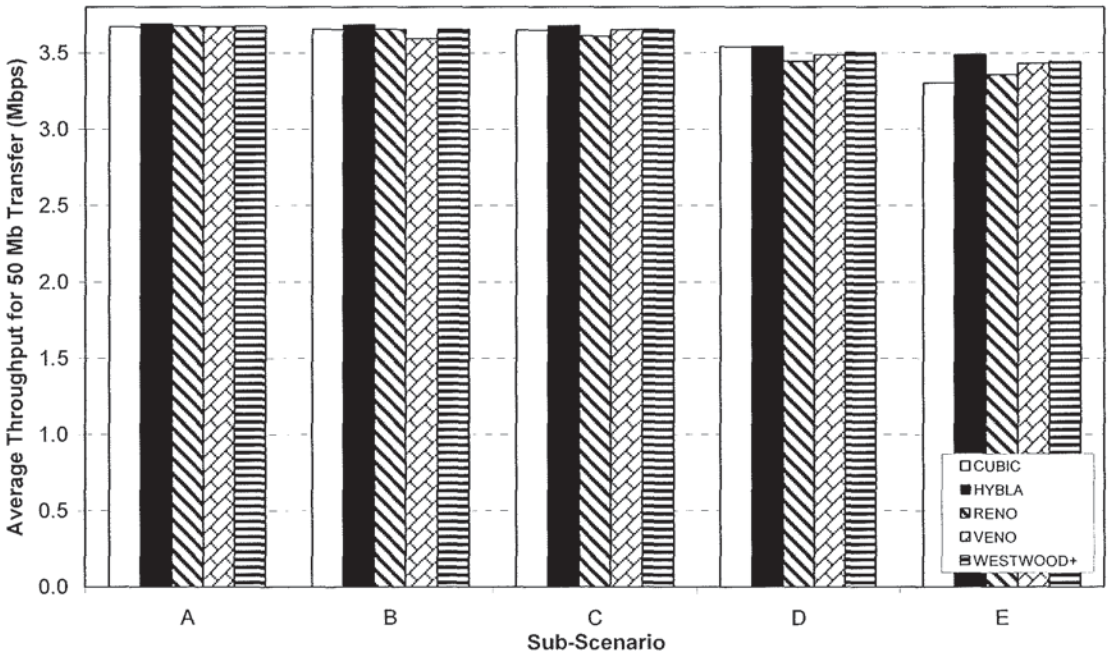


Figure 7.24: Large TCP flows: Average throughput achieved by 802.11g end-user

Looking at Figure 7.23, TCP Veno experienced a slight increase in the average transfer time in sub-scenario B, but it wasn't significant because in sub-scenario C its performance was on terms with the other algorithms. Looking at the performance of each algorithm from sub-scenario C onwards, it can be seen that Reno's performance degradation was the most accelerated. In contrast, Hybla achieved the highest throughput and quickest transfer times across all sub-scenarios.

An interesting observation is that in sub-scenario E (with five 802.11g devices in the WLAN), CUBIC was the worst performer, possessing a transfer time in excess of 127 seconds, taking over 2 seconds longer than Reno to complete the transfer, and 7 seconds longer than Hybla, which is a significant difference from an end-user's perspective. A possible explanation for this is that CUBIC has not been designed for wired-to-wireless paths, and although its performance has been better than Reno's with fewer 802.11g devices in the WLAN. This was more likely to be due to the cubic growth of its *cwnd*, which has been designed to initially possess a 'steady state' behaviour, then with further time elapsing, to possess more of a bandwidth probing behaviour [286]. In sub-scenario E, the transfer times are the longest overall, and this may have put CUBIC's *cwnd* growth into the second bandwidth probing phase. Although this probing technique may work well for CUBIC in a wired network, in wired-to-wireless paths, a sudden accelerated opening of the *cwnd* would cause large numbers of TCP data segments to arrive at the AP for transmission over the downlink. This can lead to two possible events: 1) the AP's incoming buffer queue size could overflow, causing TCP data segments to be lost completely, or 2) a large transmit queue at the AP will cause increased and variable delays for the return of TCP ACKs back to the sender, leading to RTO events at the server. Both events could be an explanation for the degraded performance of CUBIC in sub-scenario E.

From looking at Figure 7.23, it can be seen quite clearly that in sub-scenario E, Hybla, Westwood+, and Veno offer better transfer times than Reno and CUBIC. This confirms that the enhancements made in Hybla, Westwood+, and Veno for wired-to-wireless paths do indeed produce better performance than those designed for traditional wired paths only. The results indicate that the differences between these two categories of algorithms are noticeable when the last-hop WLAN consists of at least

three active 802.11g devices, and where one of the devices is downloading large TCP files from the Internet, possibly via the FTP protocol.

7.5.4 Retransmission Timeout (RTO) Timer

In this subsection the results of TCP sender's average RTO timer values are presented (obtained from the web100 kernel), which have been averaged over the duration of each data transfer, and then over the three independent trials that were conducted. As discussed earlier, the RTO timer value is an interesting area of study for researchers, as it governs the start-stop behaviour of a TCP sender in the presence of random and variable delays that are typical of last-hop WLANs.

Figure 7.25 plots the average RTO timer values for the *small* data transfers; with the exception of the legacy TCP Reno, it can be seen that the average value of the RTO timer was fairly constant for all sender-side algorithms. The average end-to-end RTT across all iperf runs was calculated to be 175.7 ms for all five sub-scenarios (from web100 statistics). For Reno then, it can be seen that its RTO timer value is quite large across all sub-scenarios in relation to the average RTT. In contrast, the other TCP algorithms seem to all possess much lower RTO timer values, implying that their response times would be better in the presence of actual TCP losses over the WLAN. Recall from Chapter 3 that a large RTO timer value causes a sender to wait for unnecessary longer periods of time whilst waiting for a TCP ACK to arrive from the WLAN. The likelihood of TCP traffic being lost over the WLAN is higher than wired networks, and so the legacy Reno is waiting on average up to three times longer than the other algorithms just to confirm such losses. Figures 7.19 and 7.20 also support this theory, which clearly show Reno to be the worst performer for small TCP transfers between sub-scenarios C to E, where channel conditions over the WLAN were more erroneous according to Figure 7.3. Hence, Reno's over-inflated RTO timer computation may be the explanation to date for its degraded performance over wired-to-wireless paths.

Figure 7.26 plots the average RTO timer values for the *medium* sized TCP transfers. It can be seen in Figure 7.26 that TCP Reno shows great variation in the value of its RTO timer, experiencing a sudden increase from sub-scenario D onwards. The average end-

to-end RTT across all iperf runs was calculated to be 184.7 ms for all five sub-scenarios. Comparing all the algorithms, it appears that Reno's computation of the RTO timer is highly sensitive to conditions over the last-hop WLAN, which again may explain its poor performance in such conditions, as can be seen in Figures 7.21 and 7.22.

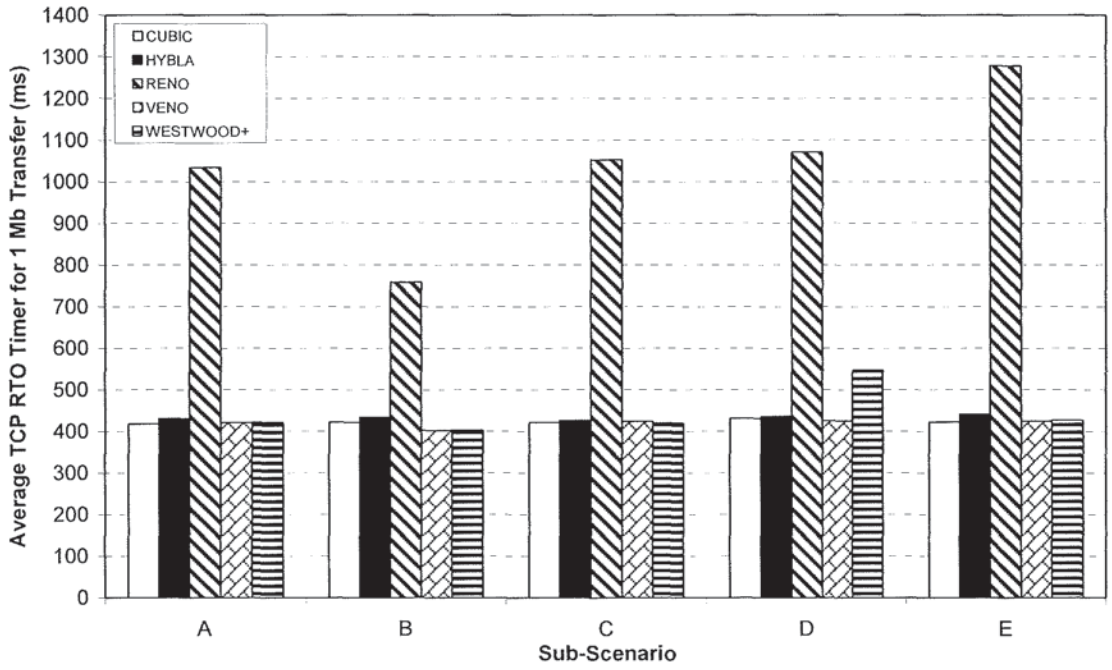


Figure 7.25: Small TCP transfer: Average RTO timer value at TCP sender

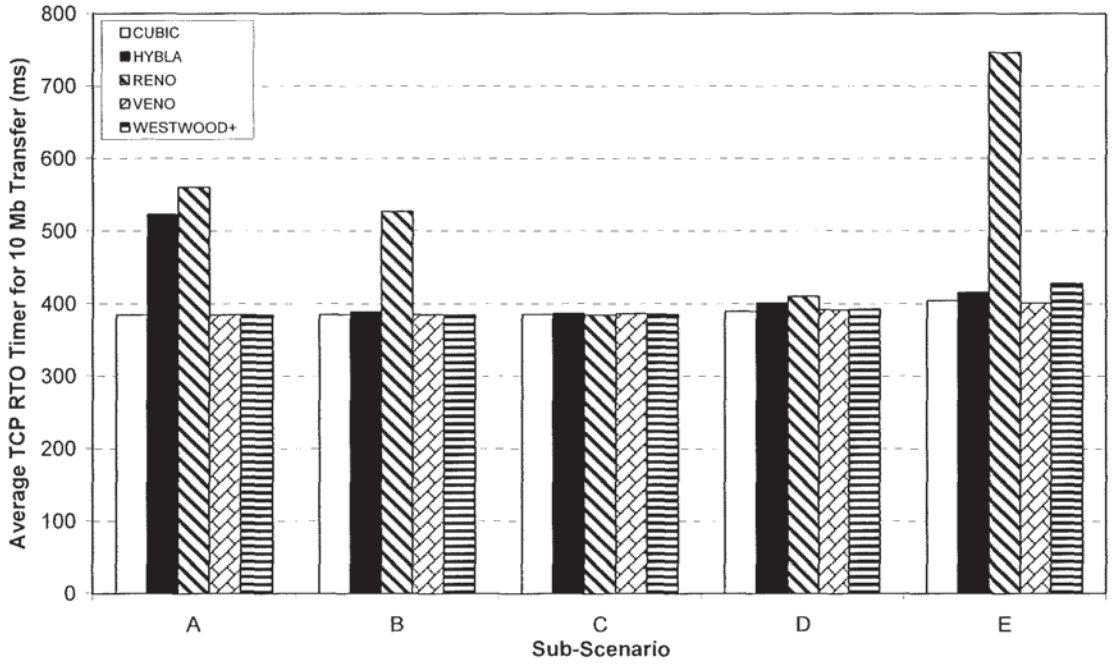


Figure 7.26: Medium TCP transfer: Average RTO timer value at TCP sender

Figure 7.27 plots the average RTO timer values for the *large* size TCP transfers. The average end-to-end RTT across all iperf runs was calculated to be 181.2 ms for all five sub-scenarios. First and foremost, Figure 7.27 shows that TCP Reno's RTO timer value appears to exhibit similar behaviour to that seen in the *small* and *medium* sized TCP transfers, again highlighting its high variability and sensitivity over such conditions. The other sender-side algorithms exhibited less variability, but generally (specifically for the *large* transfers) all of them showed an increase in the values of their RTO timer as the number of 802.11g devices in the last-hop WLAN is increased from sub-scenario A to E. Ideally, the TCP sender's RTO timer value should remain fairly constant across the sub-scenarios, or even be reduced when conditions over the WLAN degrade. The justification for this is that one would expect channel conditions over the WLAN to be the most erroneous in sub-scenarios D and E; hence TCP segments (both data and ACKs) are more likely to experience actual losses due to the 802.11 ARQ mechanisms reaching its retry limit. For this reason, it seems logical for a TCP sender to reduce the magnitude of its RTO timer as opposed to increasing it, thereby making it more responsive to the non-arrival of ACKs from the wireless end-device. Further investigations would be needed to assess the impact of reducing the RTO timer with increasing 802.11g device numbers in the last-hop WLAN on end-to-end performance of TCP congestion control algorithms.

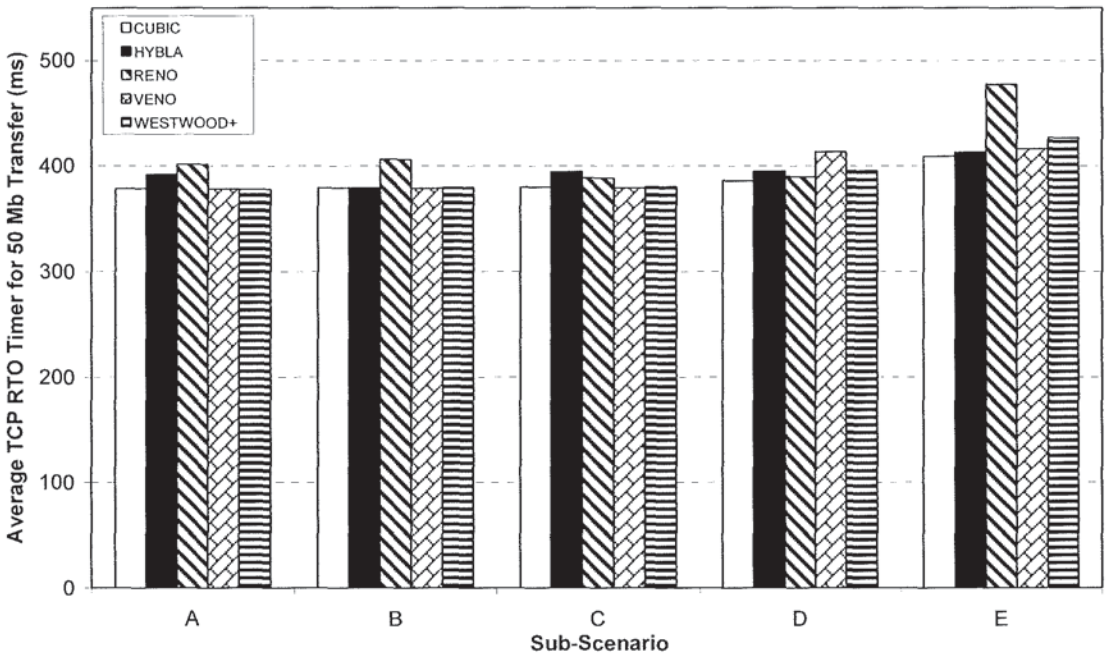


Figure 7.27: Large TCP transfer: Average RTO timer value at TCP sender

7.6 Scenario 2 – Results and Discussions

This section presents the results of TCP experiments consisting of transferring 30 Mb of data (using iperf) from the server in the wired domain to SEAMOSS in the WLAN. In scenario 2, the locations of SEAMOSS were varied around the home to create sub-scenarios of the channel conditions (signal quality) it perceived. Three sub-scenarios were chosen, representing good (SNR = ~30 dB), fair (SNR = ~20 dB), and poor (SNR = ~10 dB) signal qualities. In each sub-scenario, the wireless end-user performed the equivalent of downloading a 30 Mb file from a TCP server in the Internet. The transfer for each sub-scenario was performed three times back-to-back using each of the five congestion control algorithms on test. This section therefore presents the averages (arithmetic mean) of the three runs per sub-scenario per TCP variant.

7.6.1 Transfer Time Performance for 30 Mb Download

Figure 7.28 plots the 30 Mb transfer time performance of the five sender-side congestion control algorithms on test. The transfer time is the total download time for acquiring all the data, as reported by the iperf tool on SEAMOSS. Refer to Figure 7.4 for information relating to the frame error conditions of the WLAN in each of the three sub-scenarios.

It can be seen from Figure 7.28 that in good channel conditions all five algorithms completed the transfer within a respectable 84 seconds. Even in fair channel conditions, with the exclusion of TCP Reno, all algorithms completed the transfer within 103 seconds, with Reno taking almost 121 seconds. CUBIC completed the transfer in less than 92 seconds, showing that it was little affected by a 10 dB drop in channel conditions of the last-hop WLAN. Moving onto the poor channel conditions, TCP Hybla was the best performer here, completing the transfer within a respectable 137 seconds, with CUBIC managing to complete it within 150 seconds. In contrast, Reno took almost 300 seconds to complete the transfer, which is more than double the time taken by Hybla and CUBIC. Veno performed quite well in poor channel conditions, completing the transfer within 175 seconds, however Westwood+ took almost 238 seconds.

In summary, CUBIC and Hybla were the strongest performers across the three sub-scenarios, with Hybla experiencing only a 66% increase in the transfer time between good to poor channel conditions. In contrast, Reno experienced a 253% increase, and Westwood+ experienced a 186% increase.

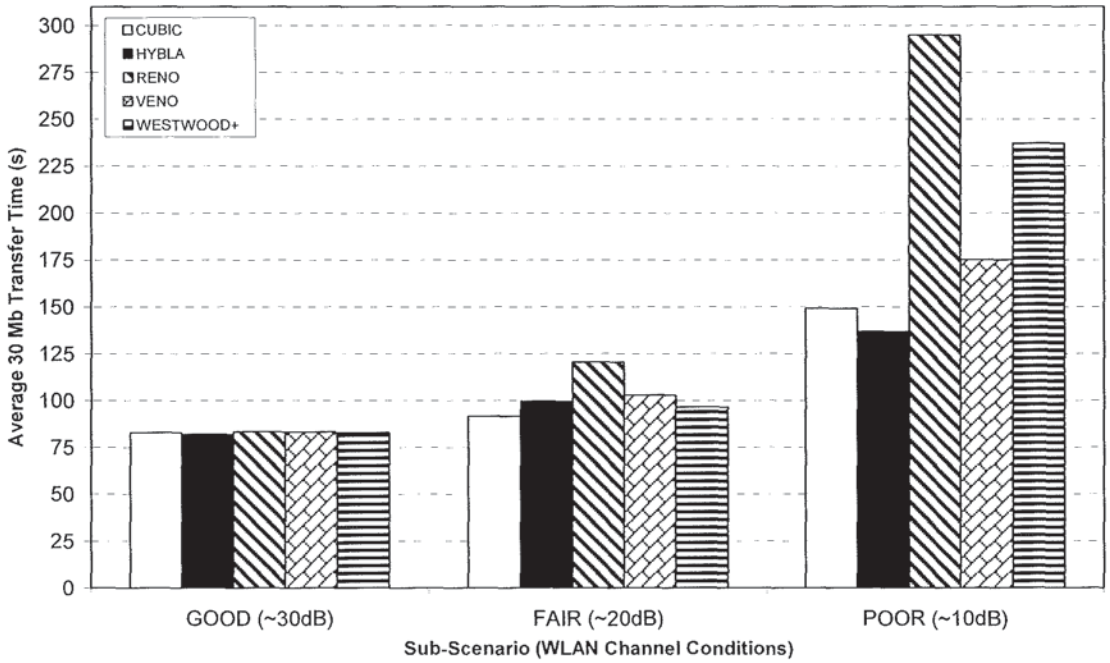


Figure 7.28: Download time for 30 Mb of data by 802.11g end-user

7.6.2 End-User Throughput Performance

In the presence of the downlink and uplink WLAN error conditions reported in Figure 7.4 for each of the sub-scenarios, Figure 7.29 plots the average achieved throughput as perceived by the end-user. The methodology described in subsection 7.5.1 using Eq. 7.8 was used to calculate the end-user throughputs.

As can be seen from Figure 7.29, in good channel conditions the maximum throughput achieved was 3.08 Mbps (Hybla), with the 3.03 Mbps (Reno) being the minimum. These values are governed by the upstream and downstream bandwidth settings of the Internet emulator of the testbed. Also governing them are the existence of FERs over the WLAN (refer to Figure 7.4).

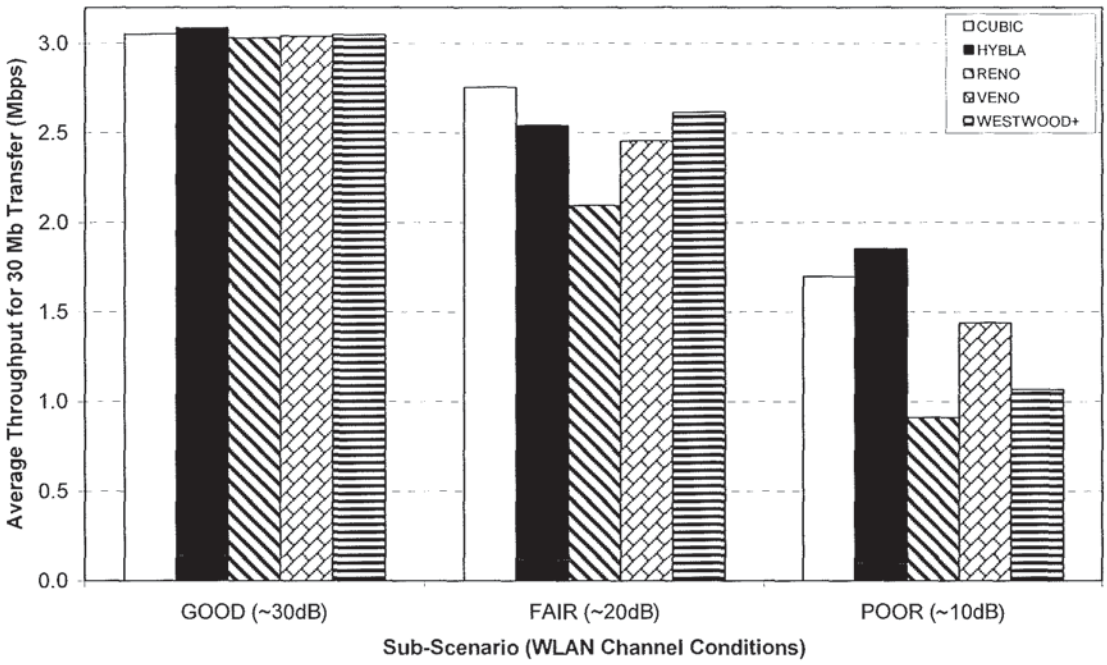


Figure 7.29: Throughput achieved by 802.11g end-user over varying WLAN conditions

Looking at the performance of the legacy Reno across the three sub-scenarios, it can be seen that it was consistently the weakest performer. Unable to deal with the high uplink and downlink FERs and transmission delays (Figure 7.12) in fair and poor channel conditions, it achieved an average throughput of just 0.9 Mbps. In contrast, Hybla and CUBIC were able to utilise more of the available WLAN bandwidth in poor channel conditions, achieving throughputs of 1.85 Mbps and 1.7 Mbps, respectively. Even Veno achieved a throughput for the end-user of 1.45 Mbps. The three algorithms were able to achieve higher throughputs due to the enhancements made at the sender-side. CUBIC has been designed to probe for more bandwidth quickly, which worked to its advantage. Veno has been designed to be less aggressive with the reduction of its *cwnd* size when losses are deemed to be random (i.e. over the WLAN), again working to its advantage. Hybla has been designed to take advantage of higher RTT paths by accelerating the growth of its *cwnd* size relative to the RTT, which (according to Figure 7.12) would have been the case due to the higher transmission delays for TCP data segments and ACKs over the WLAN.

TCP Westwood+ performed only marginally better than Reno, achieving an average end-user throughput of just 1.05 Mbps, which is surprising since its authors claim that it achieves good performance over wired-to-wireless paths with random losses. The

reason for its lacklustre performance may be due to inefficiencies with its end-to-end BWE function, which relies on the rate of returning TCP ACKs from the wireless domain. In poor channel conditions, the uplink FER was almost 30%, with average TCP ACK transmission delays of almost 8 seconds in comparison to just over 2 seconds for TCP data segments (refer to Figure 7.12). Such uplink error conditions may not have been factored into the original design of TCP Westwood+, hence forcing it to perform poorly in such real-world tests.

In summary, Hybla and CUBIC were the best performers across all channel conditions in the last-hop WLAN.

7.6.3 Retransmission Timeout (RTO) Timer

Figure 7.30 plots the TCP sender's average RTO timer values (obtained from the web100 kernel), which are averages over the duration of each data 30 Mb transfer, and then over the three independent trials that were conducted.

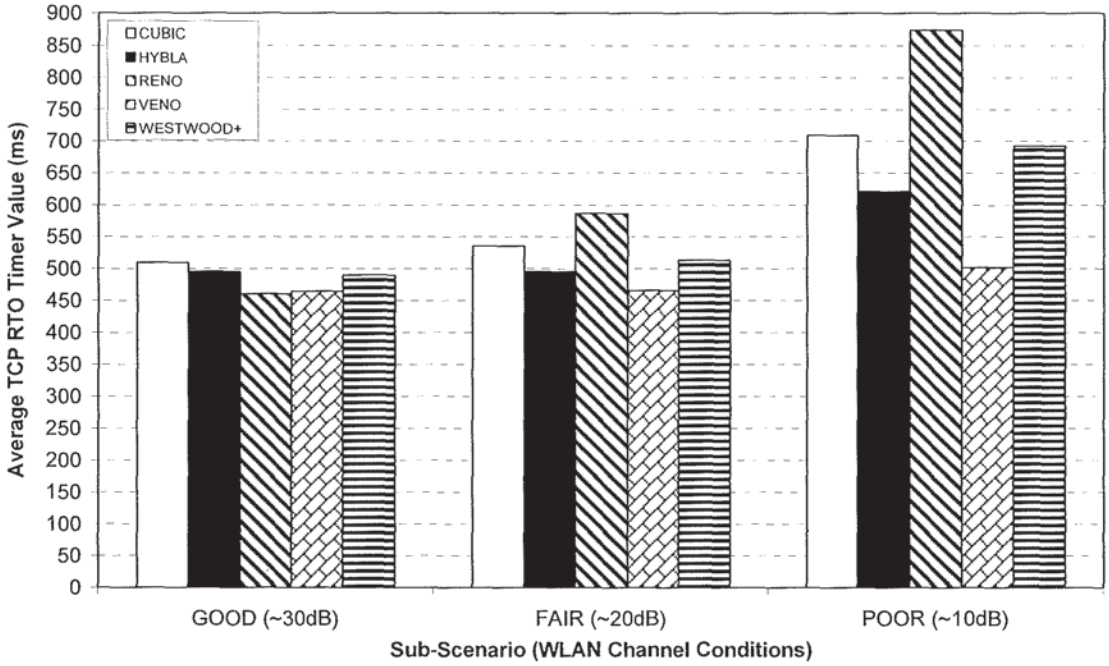


Figure 7.30: Average RTO timer value at TCP sender over varying WLAN channel conditions

Looking at Figure 7.30, TCP Reno seems to have inflated its RTO timer the most on average as the channel conditions worsen from good to poor. This increased sensitivity

follows on from the results in subsection 7.5.4, where Reno's RTO timer also exhibited similar behaviour in sub-scenarios D and E, the most erroneous conditions.

An interesting observation from Figure 7.30 is that TCP Veno maintained a relatively stable RTO timer value on average across all three channel conditions. All other algorithms increased their average RTO timer significantly only when channel conditions became poor. However, the maximum average RTO timer value seen is 873 ms (for Reno). This may have worked against all five algorithms in poor channels conditions. Looking at Figure 7.12 for poor conditions, the average delay for a TCP data segment and TCP ACK pair over the WLAN was in the region of 8000 ms, nearly ten times greater than the average RTO timer value. This implies that in poor channel conditions with significantly high FERs, a TCP sender is more likely to suffer from RTO events as opposed to fast retransmit and fast recovery events caused by triple DUPACKs. A RTO event always initiates the slow start algorithm after the retransmission of a data segment, with the *cwnd* size reduced to its initial start size (usually equal to 1 MSS or 2 MSS). This has a severe impact on the sending rate, and on the maximum end-to-end throughput that can be achieved.

7.6.4 Retransmission Behaviour at TCP Server

Figure 7.31 plots the total number of TCP data segment retransmissions that occurred at the TCP server during each 30 Mb transfer (obtained from the web100 kernel), either due to RTO events or due to the reception of three DUPACKs. The results have been averaged over the three independent transfers per sender-side algorithm, and per each channel condition.

In Figure 7.31, it can seen that in good channel conditions, on average there were no retransmissions of data segments for all sender-side algorithms, except for TCP Reno who had to retransmit a data segment just the once on average. This indicates that a TCP sender in the wired domain is able to resist FERs over a last-hop 802.11 WLAN of up to 10% for downlink traffic and up to 5% for uplink traffic (refer to Figure 7.4).

When FERs increased over the WLAN in fair conditions to over 40% for downlink traffic and over 10% for uplink traffic, the TCP sender first showed signs of

retransmissions taking place. In Figure 7.31, it can be seen that in fair conditions TCP Reno retransmitted, on average, 20 data segments, which is nearly twice as many as the other algorithms. In poor conditions, Reno retransmitted, on average, 68 data segments, which is more than triple the number of segments retransmitted by CUBIC, Hybla, and Veno. As discussed above in subsection 7.6.2, Westwood+ suffered from weak end-user performance in poor channel conditions, and Figure 7.31 reveals that it also retransmitted 52 data segments on average, which may have been due to lost TCP ACKs. Unfortunately, TCP ACKs are not retransmitted; it is always the data segment that is retransmitted by the sender. It may be better to perhaps incorporate a heuristic mechanism at the sender for determining that a TCP ACK has been lost, and hence a retransmission of the data segment is not necessary, thereby preventing a reduction in the sending rate.

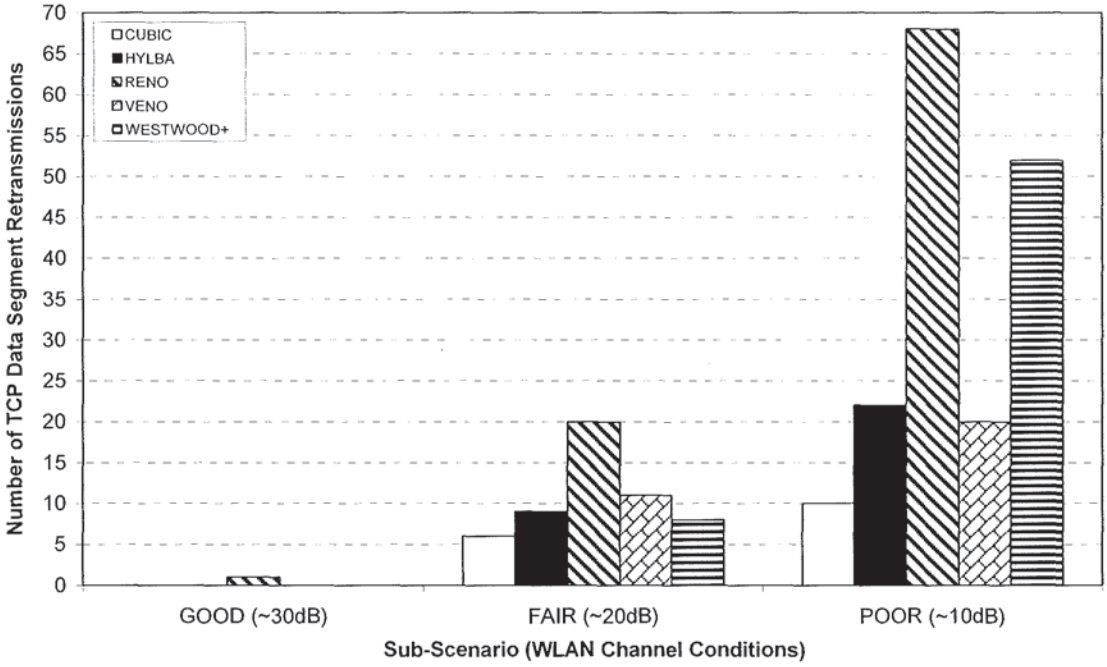


Figure 7.31: Retransmissions by the TCP sender over varying WLAN channel conditions

7.7 Chapter Conclusions

This chapter presented experimental results evaluating the performance of TCP Reno, CUBIC, Hybla, Veno, and Westwood+ sender-side congestion control algorithms over wired-to-wireless paths, using a variety of scenarios that accurately replicated the real-

world. TCP Reno is the legacy algorithm used by servers in the Internet (and most likely still being widely used); hence it was included in the tests in order to obtain some base-line results. CUBIC is the default algorithm used by the latest Linux v2.6 kernels upon installation; hence it was included in the tests due to the popularity of Linux as an operating system on many of the servers in the Internet today. TCP Hybla, Veno, and Westwood+ are all specific (and recent) modifications to the legacy TCP AIMD algorithm in an attempt to provide enhanced performance over wired-to-wireless paths; hence all three were chosen to be put on test in such conditions in order to verify their realistic performance and capabilities using a real-world based experimental environment. Overall, the aim of the chapter was to explore advanced experimental approaches using real-world protocols and devices in order to confirm achievable performance of recently proposed sender-side TCP algorithms in relation to their respective authors' claims.

A purpose-built wired-to-wireless experimental testbed was used extensively, which consisted of real-world implementations of the TCP algorithms being tested, an emulated fixed network domain, and a real-world operational last-hop 802.11g home WLAN with up to five devices. Prior to the actual experimental work with the various TCP variants, thorough investigations of the transmission and error characteristics of the WLAN for each of the main scenarios (1 and 2) and their respective sub-scenarios were conducted using the full capabilities of the testbed. This was to provide further insights of conditions of a typical home-office WLAN, as well as to support the results generated by the remainder of the work in the chapter. After vigorous testing work and evaluations, the key concluding remarks based on the experimental results achieved have been summarised below:

1. It was discovered that, with the notable exception of TCP Reno, each of the recent sender-side TCP proposals performed on comparable terms, with Hybla consistently being the best performer in all of the testing sub-scenarios and for all transfer sizes (scenario 1). Note that Hybla was specifically designed for high latency wired-to-wireless paths, using a variety of techniques aimed specifically at enhancing the legacy AIMD mechanism. Its superiority in such conditions has been confirmed, outperforming the other wired-to-wireless enhancement proposals, Veno and Westwood+.

2. It was observed from all experiments of scenario 1 that there was a clear turning point in the data transfer performance for all TCP algorithms when the number of 802.11g devices in the last-hop WLAN exceeded three. For each additional 802.11g device that was added to the WLAN, the TCP performance were reduced; this includes *small*, *medium*, and *large* sized data transfers from the wired domain. However, for up to three WLAN devices, TCP performance degradation was minimal. Using greater than three 802.11g devices in the WLAN appeared to have affected end-to-end TCP performance more noticeably, although it could be argued that the turning point could easily have been due to peculiarities with the third end-device (i.e. TEMPLETREE) being introduced into the experiments.

3. With regards to the legacy TCP Reno, the most likely congestion control algorithm to still be used by servers in wired domains, it was noticed that it was actually the worst performer in the majority of the tests for both scenario 1 and scenario 2. This fully supports the many claims researchers have made about its poor performance over wired-to-wireless conditions.

4. From further observations into the RTO timer values for each algorithm tested, TCP Reno revealed some interesting insights that may explain its poor performance in wired-to-wireless paths, which may have been overlooked in previous work. It was learnt that Reno's computation of the RTO timer during a connection was much more sensitive to changes in the last-hop WLAN. As the sub-scenarios of scenario 1 and scenario 2 in the WLAN were varied, Reno's RTO timer computations exhibited high variability from sub-scenario to sub-scenario, and in some cases even showed a significant increase in its average value (in comparison to the other algorithms) when conditions were more erroneous over the WLAN. Increasing the value of RTO timer significantly when TCP losses are more likely leads to stalled growth of the sender's *cwnd* due to long waiting periods between the sending of data and the non-arrival of ACKs, which only leads to a *cwnd* reduction anyway. There is clearly a trade-off between having a large RTO timer value and a small one, which is heavily dependent on the conditions of the last-hop WLAN. Further investigations would be required to study the TCP RTO timer in more detail using the experiment testbed.

5. Interestingly, TCP CUBIC, although designed primarily for wired networks, was a strong performer in all of the transfer sizes. A simple explanation for its effectiveness over wired-to-wireless paths is that CUBIC was designed to probe for unused bandwidth via its cubic *cwnd* evolution, with the aim of maximising throughput performance. This technique seemed to have worked well over the various sub-scenarios, as the available bandwidth over the last-hop WLAN (54 Mbps) was higher than that of the wired path, whose bandwidth was governed and shaped according to the Internet emulator (refer to Chapter 6). Hence CUBIC was able to take advantage. However, it was found that for very long connection times (i.e. large sized data transfers), CUBIC reduced its aggressiveness, and was easily overtaken by the other sender-side algorithms.

6. TCP CUBIC and Hybla were the strongest candidates for large sized (30 Mb) data transfers over good, fair, and poor channel conditions in the last-hop WLAN. They performed particularly well over poor channel conditions in relation to the other algorithms.

7. The Westwood+ sender suffered from poor performance when used over poor channel conditions in last-hop WLAN, with high FERs. It caused the sender to perform a high average number of data segment retransmissions, therefore trailing not too far behind TCP Reno in such conditions.

8. Finally, it was found that the performance of Veno and Westwood+ were on very close terms throughout the tests of scenario 1 involving multiple end-devices in the WLAN, exhibiting similar performance across all sub-scenarios and data transfer sizes.

Overall, a consistent method for performing standardised testing for TCP congestion control algorithms over wired-to-wireless paths has been demonstrated, with experimental results yielding useful and accurate insights. Such insights could be a step forward in the development and/or refinement of a TCP congestion control algorithm that is able to operate with maximum efficiency over an increasingly heterogeneous Internet.

Chapter 8

Thesis Conclusions and Future Work

8.1 Summary of Conclusions

The contributions of this thesis have been two-fold; a) it has thoroughly explored and reviewed the current state of art relating to the key issues as well as some of the most prominent solutions concerning fixed TCP senders over wired-to-wireless paths, with a particular focus on those wireless paths using the increasingly popular IEEE 802.11 WLAN technology (Chapter 3), and b) it has demonstrated via comprehensive experimental work the performance issues that fixed TCP senders suffer from in a variety of conditions over last-hop 802.11 WLAN paths in the real-world environment (Chapters 4 to 7). Further challenges and issues associated with the use of TCP in conjunction with 802.11 WLANs have been identified in this thesis, the work of which has constituted as the core focus.

In Chapter 4 simulations using the popular OPNET Modeler^{TM} were performed to gauge the behaviour of a TCP sender under varying BERs over the wireless channel. A custom bit error module was implemented, which was then used extensively to assess the performance thresholds for TCP and associated applications. This chapter has successfully highlighted and confirmed the ineffectiveness of TCP in conditions where the BER over the WLAN channel is in excess of ~10^{-5}, thus affecting wireless end-users of popular Internet application-layer protocols such as HTTP and FTP. It was also revealed that HTTP traffic performance can sustain higher BERs over the WLAN than larger sized file downloads from the Internet using FTP.

In Chapter 5, the results obtained provide strong evidence that there are significant differences in the sending performance of TCP between differing path-error conditions, regardless of the sender-side TCP variant used. The significant impacts of excluding a loss model on the reverse channel for TCP experiments were highlighted through a series of controlled experiments over an emulation platform using real-world

implementations of TCP. The general conclusions from this work are that in experiments where losses are induced only on the forward data channel, the performance results for TCP are easily over-estimated. In reality, over wireless channels, TCP experiences losses in both directions; hence ACKs on the reverse channel are also affected. Such bidirectional losses were induced into the experiments of this chapter, and the performance of TCP was shown to be worse-off. The message to researchers of TCP issues for wireless networks is that all experiments should be configured to use forward and reverse channel loss models in order to fully justify the capabilities of the protocol.

In Chapter 6 a purpose-built testbed for conducting more advanced and accurate experiments using TCP over last-hop 802.11 WLANs is proposed. A strong feature of the testbed is that it can be constructed from readily available hardware and software tools, encouraging researchers to add new dimensions to their experimental works. The importance of an Internet component within wired-to-wireless TCP experiments has also been stressed, being a key part of the testbed. Typical path characteristics of the Internet were also gauged, which may be useful to other researchers in the area. The uniqueness of the testbed is that it can be used to study the TCP-layer of a sending machine in the wired domain when there is an Internet and a real-world wireless path in the journey of TCP segments and ACKs, which is difficult to do in the real-world. In addition to studying TCP behaviour, the testbed allows researchers to conduct detailed studies into transmission behaviour of the IEEE 802.11 MAC. The novelty lies with the fact that the testbed can simultaneously capture data/information from the TCP-layer and from over the 802.11 WLAN from all angles of the same TCP experiment, from both the sender's and the receiver's perspective. Having such a wealth of captured data becomes invaluable when evaluating the end-to-end performance of TCP over wired-to-wireless paths.

To demonstrate the effectiveness of the testbed, a wide set of results from the wired-to-wireless testbed confirmed that 802.11 frame errors are prevalent over WLAN channels in both downlink and uplink channels. But interestingly it was also discovered that the reverse channel of the WLAN possessed its own frame error behaviours, quite different from forward channel error rates. Further observations led to the conclusion that there may be more correlation between reverse channel error

rates over the WLAN and the number of times a TCP sender invokes its retransmission mechanism, more so than for forward channel FERs.

Also investigated in Chapter 6 were the probability distributions of the number of times a particular 802.11 data frame is retransmitted by the AP and by the 802.11 end-device; this again highlights the statistics of data that can be extracted from the testbed. The results show that when channel conditions degrade, an 802.11 AP and end-device will maximise the use of their retransmission mechanism at the MAC. Such real-world observations may be useful to those working with error correction techniques to design optimal codes based on accurate channel behaviours, or equally to those wishing to gain an understanding of how TCP's RTO timer could be closely aligned to the highly variable delay conditions occurring over an 802.11 WLAN due to localised retransmissions. The aim was to inspire future work that looks at 'cross-layer' relationships between fixed TCP senders and the characteristics of an 802.11 WLAN.

Chapter 7 presented experimental results evaluating and comparing the performance of TCP Reno, CUBIC, Hybla, Veno, and the Westwood+ sender-side congestion control algorithms that are present in the recent Linux v2.6 kernel over a wired-to-wireless path, using a variety of scenarios that accurately replicated a real-world home-office WLAN environment. The proposed experimental testbed from Chapter 6 was used extensively, which consisted of real-world implementations of the TCP algorithms being tested, an emulated fixed network domain, and a real-world operational last-hop 802.11g home WLAN with up to five devices. After vigorous experimental work, a summary of the concluding remarks are listed below:

- It was discovered that, with the notable exception of TCP Reno, each of the recent sender-side TCP proposals performed on comparable terms, with Hybla consistently being the best performer in all of the tests.

- It was observed that there was a clear turning point in the data transfer performance for all TCP algorithms when the number of 802.11g devices in the last-hop WLAN exceeded three. For each additional 802.11g device that was added to the WLAN, the TCP performance were reduced. However, for up to three WLAN devices, TCP performance degradation was minimal.

- It was noticed that TCP Reno was actually the worst performer in the majority of the tests. This fully supports, through real-world tests, the many claims that have been made about its poor performance over wired-to-wireless conditions.

- It was discovered that TCP Reno's computation of the RTO timer during a connection was much more sensitive to changes in the last-hop WLAN. Reno's RTO timer computations exhibited high variability, and in some cases even showed significant increases in its average value (in comparison to the other algorithms) when conditions were more erroneous over the WLAN.

- Interestingly, TCP CUBIC, although designed primarily for wired networks, was a strong performer in all of the transfer sizes.

- TCP CUBIC and Hybla were the strongest candidates especially for large sized (30 Mb) data transfers over good, fair, and poor channel conditions in the last-hop WLAN. They performed particularly well over poor channel conditions in relation to the other algorithms.

- The TCP Westwood+ sender was a weak performer when used over poor channel conditions in last-hop WLAN, where the FERs were high. It caused the sender to have a higher average number of data segment retransmissions, therefore trailing not too far behind TCP Reno in such conditions.

- It was found that the performance of TCP Veno and Westwood+ were on very close terms throughout the tests involving multiple end-devices in the WLAN, exhibiting similar performance across all data transfer sizes.

8.2 Summary of Contributions

8.2.1 A New Synthesis of TCP Issues over IEEE 802.11 Wireless Paths

The thesis has made a new synthesis of the current state of the art based on available literature and proposals relating to the performance issues of TCP senders over wired-to-wireless paths. A vast collection of recent literature has been digested and presented in a form that can be easily interpreted, allowing a future reader to quickly identify key problem areas of interest, as well as the most prominent proposals and authors that correspond to the area. To elaborate, this thesis has identified the most prominent developments and key challenges in the area (Chapters 2 and 3), presenting the ideas in the form of a concise and systematic survey. In summary, the systematic survey is a collection of critical and constructive viewpoints based on a comprehensive review of related and recent work/innovations in the area.

8.2.1 A CBG Module for the WLAN Model in OPNET Modeler™

Since the majority of researchers use the NS-2 simulator to study TCP over wireless paths, this thesis opted to use the more commercial OPNET Modeler™ simulation suite as an alternative, comprising of very accurate models of TCP and the IEEE 802.11 protocols. To capture the impacts of varying BERs over the WLAN a *custom BER generator* (CBG) was developed to replace the standard BER stage of the radio transceiver pipeline (RTP) in OPNET Modeler™, i.e. *stage 11* was replaced. The CBG module was also contributed to the OPNET Modeler™ source-code community.

8.2.2 Significance of Bidirectional Error Models for TCP Experiments

The significance of ensuring that all TCP experimental work undertaken should include a loss model for both data segments on the forward channel and ACKs on the reverse channel was shown. This is to prevent an over-estimation of the capabilities of TCP senders over such conditions.

8.2.3 A Platform for TCP Experiments over Wired-to-Wireless Paths

A wired-to-wireless testbed has been proposed, alongside a full demonstration of how to perform effective TCP experiments over wireless paths, with a particular focus on the IEEE 802.11 WLAN technology. Various software tools and new techniques have been proposed, which have not previously been shown. The use of all the components allows the capturing of the full picture of the interactions between the TCP protocol and the IEEE 802.11 MAC protocol in unison for the same experiment.

8.2.4 Real-World Error Characteristics of IEEE 802.11 WLANs

Clear and accurate insights into the characteristics of a real-world home-office 802.11 WLAN have been presented, the results of which can be used in future research to minimise assumptions. Firstly, a novel methodology is proposed for calculating the independent frame error rates (FERs) for downlink and uplink TCP traffic flows over an 802.11 WLAN. Secondly, a new technique for obtaining the distribution of retransmission probabilities associated with the 802.11 MAC ARQ mechanisms of both the AP and the end-device is presented. Thirdly, a new technique for measuring the delays experienced by individual TCP data segments in the downlink direction and by TCP ACKs in the uplink direction of the WLAN is presented, again with useful results obtained. The novelty lies in the fact that all three techniques can be applied to the real-world captures.

8.2.5 Evaluating Linux v2.6 TCP Variants over Wired-to-Wireless Paths

This thesis has explored and evaluated for the first time implementations of various sender-side TCP variants in the Linux v2.6 kernel over wired-to-wireless conditions, using the proposed testbed consisting of a real-world wireless path. Five variants were evaluated and compared against each other using a systematic and fair approach. Although there were winners and losers based on the achieved results, the intentions of this work ultimately is to encourage researchers to adopt new and more accurate techniques for experimenting with TCP over wired-to-wireless paths. Specifically, the use of Linux across servers in the Internet is gaining popularity and it is therefore of paramount importance in understanding how to study the various TCP congestion control implementations on offer. This will ensure that the strongest performers will

remain in the kernel as it evolves, leading to a better quality of service in the long-run for the increasing masses of wireless last-hop Internet users who will rely on TCP for the foreseeable future at least.

8.3 Directions for Future Work

The work presented in this thesis has explored an interesting avenue regarding the future of TCP in the heterogeneous Internet. It should be reemphasised that the focus consistently has been on those TCP senders in the Internet that communicate with wireless 802.11 end-users situated at the edges day-in day-out, where the majority of the path is wired, with the last-hop path being wireless. In addition, any changes or enhancements to TCP for wired-to-wireless should be limited to the TCP implementation at the sender-side in the wired domain, i.e. at servers in the Internet.

To further develop and build on the work undertaken in this thesis (in accordance with the above vantage point) there are several directions of future work suggested, which have been listed below:

- Although the simulation work undertaken in OPNET Modeler™ was sufficient enough to investigate the effects of channel BERs over the WLAN on TCP performance, the work could be extended by implementing a more accurate channel error model that better reflects the real-world. The CBG module that was developed could be replaced with a channel error model that involves more stages of the RTP. It is suggested that the BER over the WLAN channel should be determined according to the SNR perceived by an 802.11 end-device, which in turn is used in accordance with a modulation curve implementation that is a replication of a real-world 802.11bg adapter based on its data sheet. To enhance realism further, it is suggested that the SNR perceived by the 802.11 end-devices is randomly fluctuated between an upper and lower bound defined to ensure the BER is not kept constant.

- The proposed testbed in this thesis uses an emulated Internet component to subject all forward and reverse traffic to conditions that would typically be experienced across the real Internet. A suggested extension to the testbed is to incorporate the real Internet between the experimental TCP server and the real-world last-hop

WLAN. One solution is to use two separate sites that are under full control of the researcher, which may be slightly challenging. Another solution is to use a 'loop-back' configuration from the same site, whereby all TCP traffic to/from the experimental server is routed out onto the Internet, traverses a wider area network, and is then re-routed back to the same site directly to/from the 802.11 WLAN.

- This thesis has provided many interesting insights into the interactions between a fixed TCP sender and conditions over the last-hop 802.11 WLAN. This provides a natural base for studying potential cross-layer interactions between the TCP protocol and the 802.11 MAC protocol. A possible suggestion is to use the idea of a loss differentiation algorithm (LDA) applied to a TCP sender that is specifically aimed at estimating conditions over an 802.11 WLAN, be it data segment losses or ACK losses, or even delays caused by contention issues due to an increased number of end-devices and traffic over the WLAN. By using the capabilities of the testbed to the fullest, many studies could be undertaken to assess whether or not there is a relationship between the two protocols. Extensive measurements from the testbed may even assist researchers to develop an analytical framework that can better predict TCP performance based on specific parameters/settings over the WLAN. For example, given that there are multiple 802.11g devices in a last-hop WLAN, and that the average SNR across all devices can be obtained, the average delays for TCP data segments and ACKs can be obtained, the delays and bandwidths over the fixed path can be obtained, then a single equation could be developed that can calculate the expected sending rate of a particular sender-side TCP variant for the entire wired-to-wireless path.

- In Chapters 6 and 7 there were several references to the TCP sender's RTO timer and how it could be better tuned to meet the dynamic nature of fluctuating delay conditions over an 802.11 WLAN. The aim here is to prevent unnecessary retransmissions of TCP data segments and subsequent reductions in the sender's *cwnd* size when there are sudden or spurious delays. Hence a possible avenue of work in this area could investigate the actual delays experienced by a TCP sender and how it impacts its RTT calculation based on a wide range of conditions over the 802.11 WLAN. Again, the proposed testbed and techniques could be used to perform an extensive set of experiments in order to collect a wide range of data. An

analysis of the captured data could then be carried out to assist researchers in developing a delay distribution or an RTT delay model for a TCP sender to refer to at times of unexpected delays. The distribution of 802.11 frame retransmission probabilities produced by the testbed would prove to be very useful here as they would allow the TCP sender to make calculated decisions as to the most likely cause of the unexpected delay. For example, if an unexpected delay suddenly causes a hike in the sender's RTT computation, then traditionally the sender is likely to timeout due to the RTO timer being exceeded. If then the sender could somehow detect the cause of the delay hike, then it is better able to maintain a higher sending rate by only reducing the *cwnd* size if data segment losses are deemed to have occurred in the wired path. Of course, knowing whether a TCP data segment loss or a TCP ACK loss has occurred are also important pieces of information. Again, a suggestion is to use a LDA at the sender-side in conjunction with known characteristics of the target last-hop 802.11 WLAN to determine what exactly has been lost, a data segment or a TCP ACK. If a TCP ACK has been lost then a retransmission by the sender may not be necessary.

List of Publications

R. Taank and X-H. Peng, "An Experimental Evaluation of Sender-Side TCP Enhancements for Wired-to-Wireless Paths: A Real-World Home WLAN Case Study", *Proc. of 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA '09)*, UK (Bradford), 2009.

R. Taank and X-H. Peng, "Impact of Error Characteristics of an Indoor 802.11g WLAN on TCP Retransmissions", *Proc. of 4th IEEE International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, China (Dalian), 2008.

R. Taank and X-H. Peng, "An Experimental Testbed for Evaluating End-to-End TCP Performance over Wired-to-Wireless Paths", *Proc. of 5th IEEE Consumer Communications & Networking Conference (CCNC '08)*, USA (Las Vegas), 2008.

R. Taank and X-H. Peng, "Investigation of the Effects of Feedback Channel Losses for Wireless TCP Experiments", *Proc. of IEEE 18th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '07)*, Greece (Athens), 2007.

R. Taank and X-H. Peng, "Performance Evaluation of TCP over Wireless Channel Conditions", *Proc. of 7th Annual Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet '06)*, UK (Liverpool), 2006.

Glossary of Terms

ACK (Acknowledgement)

An ACK is a transmission control packet transmitted by the receiving station as an affirmative response to the sending station. The ACK function is heavily used by error detection functions in popular transport protocols. The ACKs are numbered in coordination with data that has been received, and then sent to the transmitter.

AIMD (Additive Increase, Multiplicative Decrease)

The AIMD algorithm is a feedback control algorithm used by TCP during congestion control. Basically, AIMD represents a linear growth of a sending TCP's congestion window, combined with an exponential reduction when a congestion event takes place.

AP (Access Point)

An AP is a network device that allows wireless communication devices to connect to a wireless network or domain. The AP itself usually connects to an underlying wired network, and can relay data between the wireless domain and wired domain.

ARQ (Automatic Repeat Request)

ARQ is an error control method for data transmissions that use acknowledgments and timeouts to achieve reliable data transmission through retransmissions over an unreliable path.

Bandwidth

Bandwidth may refer to bandwidth capacity or available bandwidth in bits/s, which typically means the net bit rate, channel capacity or the maximum throughput of a logical or physical communication path in a digital communication system.

BER (Bit Error Rate)

The bit error ratio - also sometimes referred to as bit error rate, is the number of erroneous bits received divided by the total number of bits transmitted.

Congestion Control

Congestion control is concerned with controlling traffic entry into a telecommunications network, so as to avoid congestive collapse by attempting to avoid oversubscription of any of the processing or link capabilities of the intermediate nodes and networks and taking resource reducing steps, such as reducing the rate of sending packets.

Congestion Window (*cwnd*)

The *cwnd* determines the number of TCP bytes that can be outstanding (i.e. unacknowledged) at any time. This is a means of stopping the link between two endpoints from getting overloaded with too much traffic. The size of this window is calculated by estimating how much congestion there is between the two places.

CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance)

CSMA/CA is a network contention protocol that listens to the network in order to avoid collisions. If the channel is sensed "idle" then transmissions are permitted. If the channel is sensed as "busy" then transmission must be deferred.

Data Rate

The number of bits that are conveyed or processed per unit of time by a sending protocol, usually quantified in bits/s.

DCF

DCF is based on the CSMA/CA protocol. With DCF, 802.11 stations contend for access to the radio channel and attempt to send frames when there is no other station transmitting.

DUPACK (Duplicate ACK)

The difference between DUPACKs and normal ACKs is that while a normal ACK acknowledges one or more previously unacknowledged packets, a DUPACK re-acknowledges the same packet as the previous acknowledgement. DUPACKs are generated in response to packets arriving at the receiver out of order.

Encapsulation

A method of designing modular communication protocols in which logically separate functions in the network are abstracted from their underlying structures by inclusion or information hidden within higher level objects.

End-To-End

The end-to-end principle is one of the central design principles of the Internet and is implemented in the design of the underlying methods and protocols in the Internet. It states that, whenever possible, protocol operations should be defined to occur at the end-points of a communications system, or as close as possible to the resource being controlled.

Ethernet

Ethernet is a family of frame-based computer networking technologies for local area networks (LANs). It defines a number of wiring and signalling standards for the PHY-layer of the OSI networking model through means of network access at the MAC-layer and Link-layer, and a common addressing format.

FCS (Frame Check Sequence)

A frame check sequence (FCS) refers to the extra checksum characters added to a frame in a communication protocol for error detection and correction purposes.

FER (Frame Error Rate)

The ratio of data frames received with errors to total data frames received. It is used to determine the quality of a signal connection.

FTP (File Transfer Protocol)

A protocol for transferring files efficiently over the Internet.

HTTP (Hypertext Transfer Protocol)

A protocol for distributed, collaborative, and hypermedia information sharing.

IEEE 802.11

A set of standards for carrying out WLAN computer communications in the 2.4, 3.6 and 5 GHz frequency bands. They are implemented by the IEEE LAN/MAN Standards Committee (IEEE 802).

IEEE 802.11b

An expansion on the original 802.11 standard in July 1999, creating the 802.11b WLAN specification, supporting radio bandwidths up to 11 Mbps.

IEEE 802.11g

In June the 802.11g standard was ratified by the IEEE. It operates in the 2.4 GHz band (like 802.11b), but supports bandwidths up to 54 Mbps.

ISM

The industrial, scientific and medical (ISM) radio bands were originally reserved internationally for the use of RF electromagnetic fields for industrial, scientific and medical purposes other than communications. In general, communications equipment must accept any interference generated by ISM equipment.

Linux

Linux is a generic term referring to Unix-like computer operating systems based on the Linux kernel. Its development is one of the most prominent examples of free and open source software collaborations; typically all the underlying source code can be used, freely modified, and redistributed by anyone under the terms of the GNU GPL and other free licenses.

MAC-layer

The MAC data communication protocol is a sub-layer of the Data-Link-layer specified in the seven-layer OSI model. It provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multipoint network.

MSS (Maximum Segment Size)

The MSS is the largest amount of data, specified in bytes, that a computer or communications device can handle in a single, unfragmented unit.

NIC (Network Interface Card)

An item of computer hardware used to physically interface a computer to a network.

PHY-layer

The PHY-layer connects a link layer device (often called a MAC) to a physical medium such as an optical fibre or copper cable.

OSI (Open Systems Interconnection) Model

The OSI Model is an abstract description for layered communications and computer network protocol design. It divides network architecture into seven layers which, from top to bottom, are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layers.

Radiowave

Radiowaves are electromagnetic waves occurring on the radio frequency portion of the electromagnetic spectrum.

RTO (Retransmission Timeout)

A TCP RTO occurs when an ACK does not arrive within a certain timeframe. Usually, the RTO is not a fixed value, but changes to gain better network performance.

RTT (Round-Trip Time)

The elapsed time for a packet to traverse successfully across the network, and for its corresponding acknowledgement to return back to the sender.

SACK (Selective ACK)

A TCP receiver explicitly conveys to the sender which packets, messages, or segments in a stream are acknowledged (or those that have arrived successfully).

Slow Start Threshold (*ssthresh*)

A variable maintained by the TCP sender, whose value determines the point at which the slow start algorithm should terminate based on the size of the congestion window.

SNR (Signal-to-Noise Ratio)

The SNR is defined as the ratio of the signal power to the noise power corrupting the signal. In less technical terms, signal-to-noise ratio compares the level of a desired signal to the level of background noise. The higher the ratio, the less interference there is from the background noise.

TCP (Transmission Control Protocol)

A connection-oriented transport layer protocol in widespread use in the Internet, providing reliable end-to-end transmissions of data.

Throughput

Throughput is the average rate of successful message delivery over a communication channel, as experienced by the receiving endpoint.

WLAN (Wireless Local Area Network)

A WLAN connects two or more devices using a radiowave technology to enable wireless communication between devices in a limited area. It offers device mobility within a broad coverage area whilst still remaining connected to the network.

References

[1] J. B. Postel, "Transmission Control Protocol," *RFC 793*, 1981.

[2] S. Rewaskar, J. Kaur, and F. D. Smith, "A Performance Study of Loss Detection/Recovery in Real-world TCP Implementations." *IEEE International Conference on Network Protocols ICNP '07*, pp. 256-265, 2007.

[3] M. Welzl, *Network Congestion Control: Managing Internet Traffic*: John Wiley & Sons, 2005.

[4] T. Shih-Ching, L. Yuan-Cheng, and L. Ying-Dar, "Taxonomy and Evaluation of TCP-Friendly Congestion-Control Schemes on Fairness, Aggressiveness, and Responsiveness," *IEEE Network,* vol. 21, no. 6, pp. 6-15, 2007.

[5] G. Huston, "A Decade in the Life of the Internet," *The Internet Protocol Journal,* vol. 11, no. 2, pp. 7-18, 2008.

[6] S. Biaz, and N. H. Vaidya, ""De-Randomizing" congestion losses to improve TCP performance over wired-wireless networks," *IEEE/ACM Transactions on Networking,* vol. 13, no. 3, pp. 596-608, 2005.

[7] S. Farrell, V. Cahill, D. Geraghty *et al.*, "When TCP Breaks: Delay- and Disruption- Tolerant Networking," *IEEE Internet Computing,* vol. 10, no. 4, pp. 72-78, 2006.

[8] A. Medina, M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet," *ACM Computer Communication Review (CCR),* vol. 35, no. 2, pp. 37 - 52-37 - 52, 2005.

[9] V. Cerf, "A Decade of Internet Evolution," *The Internet Protocol Journal,* vol. 11, no. 2, pp. 2-6, 2008.

[10] S. Floyd, and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Transactions on Networking,* vol. 9, no. 4, August 2001, 2001.

[11] Y. Tian, K. Xu, and N. Ansari, "TCP in Wireless Environments: Problems and Solutions," *IEEE (Radio) Communications Magazine,* vol. 43, no. 3, pp. S27 - S32, 2005.

[12] F. Racaru, M. Diaz, and C. Chassot, "Quality of Service Management in Heterogeneous Networks." *International Conference on Communication Theory, Reliability, and Quality of Service CTRQ '08*, pp. 83-88, 2008.

[13] H. ElAarag, "Improving TCP Performance over Mobile Networks," *ACM Computing Surveys,* vol. 34, no. 3, pp. 357 - 374, 2002.

[14] F. Ren, X. Huang, F. Liu *et al.*, "Improving TCP Throughput over HSDPA Networks," *IEEE Transactions on Wireless Communications,* vol. 7, no. 6, pp. 1993-1998, 2008.

[15] G. Huston, "TCP in a Wireless World," *IEEE Internet Computing,* vol. 5, no. 2, pp. 82 - 84-82 - 84, 2001.

[16] S. Haykin, and M. Moher, *Modern Wireless Communications*, International ed.: Prentice Hall, 2004.

[17] M. C. Chan, and R. Ramjee, "Improving TCP/IP performance over third generation wireless networks." *IEEE INFOCOM '04*, pp. 1893-1904, vol.3, 2004.

[18] A. H. Zahran, B. Liang, and A. Saleh, "Mobility Modeling and Performance Evaluation of Heterogeneous Wireless Networks," *IEEE Transactions on Mobile Computing,* vol. 7, no. 8, pp. 1041-1056, 2008.

[19] A. C. B. Linwa, and S. Pierre, "Discovering the architecture of geo-located web services for next generation mobile networks," *IEEE Transactions on Mobile Computing,* vol. 5, no. 7, pp. 784-798, 2006.

[20] IEEE 802 Working Group 11, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Working Group for WLAN Standards*, 2007.

[21] M. Gerami, and J. Hwang, "Analysis of Broadband Diffusion in OECD Countries." *International Multi-Conference on Computing in the Global Information Technology*, pp. 11-11, 2007.

[22] B. P. Crow, I. Widjaja, L. G. Kim *et al.*, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine,* vol. 35, no. 9, pp. 116-126, 1997.

[23] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*, First ed.: O'Reilly & Associates, 2002.

[24] T. Braun, G. Carle, S. Fahmy *et al.*, *Wired/Wireless Internet Communications*: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2006.

[25] T. S. Rappaport, *Wireless Communications: Principles and Practice*: Prentice Hall, 2002.

[26] H. Balakrishnan, V. N. Padmanabhan, S. Seshan *et al.*, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Transactions on Networking,* vol. 5, no. 6, December 1997.

[27] K. Pentikousis, "TCP in Wired-Cum-Wireless Environments," *IEEE Communications Surveys & Tutorials,* vol. 3, no. 4, 2000.

[28] V. Tsaoussidis, and I. Matta, "Open Issues on TCP for Mobile Computing," *Journal of Wireless Communications and Mobile Computing,* vol. 2, no. 1, 2002.

[29] W. Stallings, *Wireless Communications and Networks*: Pearson Education, 2004.

[30] G. Xylomenos, G. C. Polyzos, P. Mahonen *et al.*, "TCP/IP Performance over Wireless Networks," *High Performance TCP/IP Networking*: Prentice Hall, 2002.

[31] A. Gurtov, and S. Floyd, "Modeling Wireless Links for Transport Protocols," *ACM SIGCOMM Computer Communication Review (CCR),* vol. 34, no. 2, pp. 85 - 96-85 - 96, 2004.

[32] U. Varshney, "The Status and Future of 802.11-Based WLANs," *IEEE Computer,* vol. 36, no. 6, pp. 102 - 105, 2003.

[33] H. Wu, Y. Peng, K. Long *et al.*, "Performance of Reliable Transport Protocol over IEEE 802.11 Wireless LAN: Analysis and Enhancement." *IEEE INFOCOM '02*, 2002.

[34] V. Vasudevan, M. Parikh, K. Chandra *et al.*, "TCP and IEEE 802.11b Protocol Performance in Indoor Wireless Channels." *IEEE Sarnoff Symposium*, pp. 257-261, March 2003.

[35] Ng, D. Malone, and D. J. Leith, "Experimental Evaluation of TCP Performance and Fairness in an 802.11e Test-bed." *ACM SIGCOMM '05*, pp. 17 - 22, 2005.

[36] D. J. Leith, and P. Clifford, "Modelling TCP dynamics in wireless networks." *Wireless Networks, Communications and Mobile Computing Conference*, pp. 906-911, 2005.

[37] Y. Jeonggyun, and C. Sunghyun, "Modeling and analysis of TCP dynamics over IEEE 802.11 WLAN." *Wireless On-demand Network Systems and Services Conference*, pp. 154-161, 2007.

[38] R. Bruno, M. Conti, and E. Gregori, "Throughput Analysis and Measurements in IEEE 802.11 WLANs with TCP and UDP Traffic Flows," *IEEE Transactions on Mobile Computing,* vol. 7, no. 2, pp. 171-186, 2008.

[39] L. Tianji, and D. J. Leith, "Buffer Sizing for TCP Flows in 802.11e WLANs," *IEEE Communications Letters,* vol. 12, no. 3, pp. 216-218, 2008.

[40] R. Jiang, V. Gupta, and C. V. Ravishankar, "Interactions between TCP and the IEEE 802.11 MAC protocol." *DARPA Information Survivability Conference and Exposition (DISCEX '03)*, pp. 273 - 282, 2003.

[41] A. D. Vendictis, F. Vacirca, and A. Baiocchi, "Experimental Analysis of TCP and UDP Traffic Performance over Infra-structured 802.11b WLANs." *COST 279, Technical Document 279 TD(04)033*, Ghent, Belgium, 2004.

[42] B. Adida, "It all starts at the server [World Wide Web and FastCGI]," *IEEE Internet Computing,* vol. 1, no. 1, pp. 75-77, 1997.

[43] M. Mathis, J. Heffner, and R. Reddy, "Web100: Extended TCP Instrumentation for Research, Education, and Diagnosis," *ACM SIGCOMM Computer Communication Review,* vol. 33, no. 3, pp. 69 - 79, 2003.

[44] M. Allman, "A Web Server's View of the Transport Layer," *ACM Computer Communication Review,* vol. 30, no. 5, 2000.

[45] S. Floyd, and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking,* vol. 7, no. 4, pp. 458-472, 1999.

[46] G. Karlsson, and I. Mas, "Quality of Service and the End-to-End Argument," *IEEE Network,* vol. 21, no. 6, pp. 16-21, 2007.

[47] A. Prasad, and N. Prasad, *802.11 WLANs and IP Networking: Security, QoS, and Mobility*: Artech House, 2005.

[48] D. Damjanovic, M. Welzl, and K. Munir, "Modern TCPs in the Internet: Survival of the Fittest." *EuroFGI Workshop on IP QoS and Traffic Control*, December 2007.

[49] W. R. Stevens, *TCP/IP Illustrated, Volume 1*: Addison Wesley, 1994.

[50] C. H. Nam, S. C. Liew, and C. P. Fu, "An experimental study of ARQ protocol in 802.11b Wireless LAN." *Wireless Personal Multimedia Communications (WPMC '02)*, October 2002.

[51] M. Franceschinis, M. Mellia, M. Meo *et al.*, "Measuring TCP over WiFi: A Real Case." *WinMee: 1st Workshop on Wireless Network Measurements*, 2005.

[52] J. S. Vardakas, I. Papapanagiotou, M. D. Logothetis *et al.*, "On the End-to-End Delay Analysis of the IEEE 802.11 Distributed Coordination Function." *Second International Conference on Internet Monitoring and Protection ICIMP '07*, pp. 16-16, 2007.

[53] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on Selected Areas in Communications,* vol. 18, no. 3, pp. 535-547, 2000.

[54] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," *IEEE/ACM Transactions on Networking (TON),* vol. 6, no. 4, pp. 485 - 498, August 1998, 1998.

[55] F. Zheng, M. Li, and C. Gao, "An analytic throughput model for TCP Reno over wireless networks." *International Conference on Computer Networks and Mobile Computing*, pp. 111-116, 2001.

[56] F. Anjum, and L. Tassiulas, "Comparative Study of Various TCP Versions Over a Wireless Link With Correlated Losses," *IEEE/ACM Transactions on Networking (TON),* vol. 11, no. 3, pp. 370 - 383, June 2003, 2003.

[57] F. Vacirca, A. D. Vendictis, A. Todini *et al.*, "On the Effects of ARQ Mechanisms on TCP Performance in Wireless Environments." *IEEE Globecom '03*, 2003.

[58] H. ElAarag, and M. Bassiouni, "Simulation of Transport Protocols over Wireless Communication Networks." *Winter Simulation Conference '02*, pp. 1235 - 1241, 2002.

[59] G. Judd, and P. Steenkiste, "Using emulation to understand and improve wireless networks and applications." *2nd ACM Symposium on Networked Systems Design & Implementation*, pp. 203-216, 2005.

[60] DARPA, NSF, and ACIRI, "The Network Simulator." www.isi.edu/nsnam/ns/

[61] OPNET Technologies. "http://www.opnet.com/."

[62] J. Heidemann, N. Bulusu, J. Elson *et al.*, "Effects of detail in wireless network simulation." *SCS Multiconference on Distributed Simulation*, pp. 3-11, January 2001.

[63] D. Kotz, C. Newport, and C. Elliot, "The Mistaken Axioms of Wireless-Network Research," *Technical Report TR2003-467*, 2003.

[64] P. Zheng, and L. M. Ni, "EMPOWER: A Network Emulator for Wireline and Wireless Networks." *IEEE INFOCOM '03*, pp. 1933 - 1942, 2003.

[65] P. Ikkurthy, J. Shahbazian, M. A. Labrador *et al.*, "Testing large scale streaming Internet applications over wireless LANs." *Eighth IEEE International Symposium on High Assurance Systems Engineering*, pp. 109-115, 2004.

[66] R. Beuran, J. Nakata, T. Okada *et al.*, "A Multi-Purpose Wireless Network Emulator: QOMET." *22nd International Conference on Advanced Information Networking and Applications - Workshops AINAW '08*, pp. 223-228, 2008.

[67] S. Guruprasad, R. Ricci, and J. Lepreau, "Integrated network experimentation using simulation and emulation." *TRIDENTCOM* '05, pp. 204-212, 2005.

[68] D. Raychaudhuri, I. Seskar, M. Ott *et al.*, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols." *IEEE Wireless Communications and Networking Conference*, pp. 1664-1669 Vol. 3.

[69] E. David, and S. Peter, "Measurement and analysis of the error characteristics of an in-building wireless network.", *Applications, technologies, architectures, and protocols for computer communications*, 1996.

[70] P. Chatzimisios, V. Vitsas, and A. C. Boucouvalas, "Revisit of fading channel characteristics in IEEE 802.11 WLANs: independent and burst transmission errors." *IEEE PIMRC* '06, pp. 1-6, 2006.

[71] S. Mukherjee, K. Jones, M. O'Shea *et al.*, "The Digital Patient Push - Using Location to Streamline the Surgical Journey." *MEDSIP '06*, pp. 1-4, 2006.

[72] B. Alexander, *802.11 Wireless Network Site Surveying and Installation*: Cisco Presss, 2004.

[73] A. Willig, M. Kubisch, C. Hoene *et al.*, "Measurements of a wireless link in an industrial environment using an IEEE 802.11-compliant physical layer," *IEEE Transactions on Industrial Electronics,* vol. 49, no. 6, pp. 1265-1282, 2002.

[74] Y. Jihwang, and A. Agrawala, "Packet error model for the IEEE 802.11 MAC protocol." *IEEE PIMRC '03*, pp. 1722-1726, vol.2, 2003.

[75] D. Malone, P. Clifford, and D. J. Leith, "MAC Layer Channel Quality Measurement in 802.11," *IEEE Communications Letters,* vol. 11, no. 2, pp. 143-145, 2007.

[76] J. Yeo, S. Banerjee, and A. Agrawala, *Measuring Traffic on the Wireless Medium: Experience and Pitfalls*, Technical Report CS-TR 4421, University of Maryland, 2002.

[77] F. Filali, "Impact of Link-Layer Fragmentation and Retransmissions on TCP performance in 802.11-based Networks." *IEEE International Conference on Mobile and Wireless Communication Networks*, September 2005.

[78] L. Pavilanskas, "Analysis of TCP algorithms in the reliable IEEE 802.11b link." *ASMTA '05*, 2005.

[79] W. Ge, Y. Shu, L. Zhang *et al.*, "Measurement and Analysis of TCP Performance in IEEE 802.11 Wireless Network." *Canadian Conference on Electrical and Computer Engineering*, pp. 1846 - 1849, 2006.

[80] H. Xie, and B. Yi, "A MAC-Layer Adaptation Algorithm Based On TCP Control Segment in Wireless LANs." *International Workshop on Cross Layer Design IWCLD '07*, pp. 105-108, 2007.

[81] T. F. Herbert, *The Linux TCP/IP Stack: Networking for Embedded Systems*: Charles River Media, 2004.

[82] P. Sarolahti, and A. Kuznetsov, "Congestion Control in Linux TCP." *USENIX '02*, 2002.

[83] I. McDonald, and R. Nelson, "Congestion Control Advancements in Linux," in *Australia's National Linux Conference (LCA '06)*, Dunedin, New Zealand, 2006.

[84] S. P. Bhattacharya, and V. Apte, "A Measurement Study of the Linux TCP/IP Stack Performance and Scalability on SMP systems." *First International Conference on Communication System Software and Middleware*, pp. 1-10, 2006.

[85] The Apache HTTP Server Project, http://httpd.apache.org/.

[86] L. Yee-Ting, D. Leith, and R. N. Shorten, "Experimental Evaluation of TCP Protocols for High-Speed Networks," *IEEE/ACM Transactions on Networking,* vol. 15, no. 5, pp. 1109-1122, 2007.

[87] V. Cerf, and Y. Dalal, "Specification of Internet Transmission Control Program," *RFC 675*, 1974.

[88] R. Braden, "Requirements for Internet Hosts - Communication Layers," *RFC 1122*, 1989.

[89] DoD, "The Internet Protocol Suite (TCP/IP)," *Defense Advanced Research Projects Agency (DARPA)*, 1983.

[90] J. Nagle, "Congestion control in IP/TCP internetworks," *RFC 896*, 1984.

[91] V. Jacobson, and M. J. Karels, "Congestion Avoidance and Control," *ACM CCR,* vol. 18, no. 4, pp. 314 - 329, 1988.

[92] D. M. Chiu, and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems,* vol. 17, no. 1, pp. 1-14, 1989.

[93] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," *End2End - Interest Mailing List*, 1990.

[94] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a congestion control Algorithm: The Effects of Two-Way Traffic." *SIGCOMM Symposium on Communications Architectures and Protocols*, pp. 133-147, 1991.

[95] L. Brakmo, O. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance." *SIGCOMM '94 Symposium*, pp. 24 - 35, 1994.

[96] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," *ACM SIGCOMM Computer Communication Review,* vol. 26, no. 4, pp. 270-280, 1996.
[97] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *RFC 2001*, 1997.
[98] V. Paxson, M. Allman, S. Dawson *et al.*, "Known TCP Implementation Problems," *RFC 2525*, 1999.
[99] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *RFC 2581*, 1999.
[100] B. Moraru, F. Copaciu, L. Gabriel *et al.*, "Practical Analysis of TCP Implementations: Tahoe, Reno, NewReno." *RoEduNet '03*, 2003.
[101] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems,* vol. 2, no. 4, pp. 277 - 288-277 - 288, 1984.
[102] S. Floyd, "Congestion Control Principles," *RFC 2914*, 2000.
[103] V. Jacobson, and R. Braden, "TCP extensions for long-delay paths," *RFC 1072*, 1988.
[104] M. Allman, and V. Paxson, "On Estimating End-to-End Network Path Properties." *ACM SIGCOMM '99*, 1999.
[105] I. Psaras, and V. Tsaoussidis, "CAM02-2: WB-RTO: A Window-Based Retransmission Timeout for TCP." *IEEE GLOBECOM '02*, pp. 1-6, 2002.
[106] N. Seddigh, and M. Devetsikiotis, "Studies of TCP's retransmission timeout mechanism." *ICC '01*, pp. 1834 - 1840 vol.6, 2001.
[107] V. Paxson, and M. Allman, *Computing TCP's Retransmission Timer, RFC 2988*, 2000.
[108] M. Allman, and A. Falk, "On the Effective Evaluation of TCP," *ACM Computer Communication Review (CCR),* vol. 5, no. 29, 1999.
[109] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *RFC 1323*, 1992.
[110] K. Fall, and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review (CCR),* vol. 26, no. 3, pp. 5 - 21-5 - 21, 1996.
[111] R. Bruyeron, B. Hemon, and L. Zhang, "Experimentations with TCP Selective Acknowledgment," *ACM Computer Communication Review,* vol. 28, no. 2, 1998.
[112] M. Mathis, S. Mahdavi, S. Floyd *et al.*, "TCP Selective Acknowledgement Options," *RFC 2018*, 1996.
[113] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 3782*, 2004.
[114] J. B. Postel, "TCP maximum segment size and related topics," *RFC 879*, 1983.
[115] S. Ubik, and P. Cimbal, "Achieving reliable high performance in LFNs." *Terena Networking Conference*, 2003.
[116] D. D. Clark, "Window and Acknowledgement Strategy in TCP," *RFC 813*, 1982.
[117] R. Prasad, and M. Ruggieri, *Technology Trends in Wireless Communications*: Artech House, 2003.
[118] M. Khosrow-Pour, *Emerging Trends And Challenges in Information Technology Management*: IGI Publishing, 2006.
[119] J. Proakis, *Digital Communications*: McGraw-Hill Higher Education, 2000.

[120] IEEE 802.11b, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band," *IEEE Working Group for WLAN Standards*, 1999.

[121] IEEE 802.11g, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Further Higher-Speed Physical Layer Extension in the 2.4 GHz Band," *IEEE Working Group for WLAN Standards*, 2003.

[122] I. Howitt, "Bluetooth performance in the presence of 802.11b WLAN," *IEEE Transactions on Vehicular Technology,* vol. 51, no. 6, pp. 1640-1651, 2002.

[123] H. ElAarag, and M. Bassiouni, "Transport Control Protocols for Wireless Connections." *IEEE 49th Vehicular Technology Conference*, pp. 337 - 341, 1999.

[124] L. Peterson, and B. Davie, *Computer Networks: A Systems Approach*: Morgan Kaufmann, 2003.

[125] G. Xylomenos, G. C. Polyzos, P. Mahonen *et al.*, "TCP Performance Issues over Wireless Links," *IEEE Communications Magazine,* vol. 39, no. 4, pp. 52 - 58-52 - 58, 2001.

[126] W. C. Y. Lee, *Mobile Communications Design Fundamentals*: John Wiley and Sons, 1993.

[127] T. V. Lakshman, and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Transactions on Networking,* vol. 5, no. 3, pp. 336 - 350-336 - 350, 1997.

[128] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The Effects of Asymmetry on TCP Performance," *ACM Mobile Networks and Applications (MONET),* vol. 4, no. 3, 1999.

[129] H. Balakrishnan, V. N. Padmanabhan, S. Seshan *et al.*, "TCP Improvements for Hetergeneous Networks: The Daedalus Approach." *35th Annual Allerton Conference on Communication, Control, and Computing*, 1997.

[130] G. T. Nguyen, R. H. Katz, B. Noble *et al.*, "A Trace-Based Approach for Modeling Wireless Channel Behavior." *28th Conference on Winter Simulation*, pp. 597 - 604, 1996.

[131] D. Eckhardt, and P. Steenkiste, "Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network." *ACM SIGCOMM '96*, pp. 243-254, 1996.

[132] L. Xiaolong, and Z. Qing-An, "Influence of Bit Error Rate on the Performance of IEEE 802.11 MAC Protocol." *WCNC '07*, pp. 367-372, 2007.

[133] P. Chatzimisios, A. C. Boucouvalas, and V. Vitsas, "Performance analysis of IEEE 802.11 DCF in presence of transmission errors." *IEEE ICC '04*, pp. 3854-3858 vol.7, 2004.

[134] D. A. Eckhardt, and P. Steenkiste, "Improving wireless LAN performance via adaptive local error control." *Sixth International Conference on Network Protocols*, pp. 327-338, 1998.

[135] C. E. Palazzi, G. Pau, M. Roccetti *et al.*, "In-home online entertainment: analyzing the impact of the wireless MAC-transport protocols interference." *International Conference on Wireless Networks, Communications and Mobile Computing*, pp. 516-521 vol.1, 2005.

[136] P. Qixiang, S. C. Liew, and V. C. M. Leung, "Performance improvement of 802.11 wireless network with TCP ACK agent and auto-zoom backoff algorithm." *Vehicular Technology Conference VTC '05*, pp. 2046-2050 vol. 3, 2005.

[137] P. E. Engelstad, and O. N. Osterbo, "Analysis of the Total Delay of IEEE 802.11e EDCA and 802.11 DCF." *IEEE ICC '06*, pp. 552-559, 2006.

[138] Y. Li, K.-P. Long, W.-L. Zhao *et al.*, "Analyzing the channel access delay of IEEE 802.11 DCF." *IEEE GLOBECOM '05*, 2005.

[139] N. T. Dao, and R. A. Malaney, "Throughput Performance of Saturated 802.11g Networks." *2nd International Conference on Wireless Broadband and Ultra Wideband Communications*, pp. 31-31, 2007.

[140] P. Barsocchi, G. Oligeri, and F. Potorti, "Packet Loss in TCP Hybrid Wireless Networks." *Advanced Satellite Mobile Systems Conference ASMS '06*, 2006.

[141] J. Tourrilhes, "PiggyData: reducing CSMA/CA collisions for multimedia and TCP connections." *Vehicular Technology Conference VTC '99*, pp. 1675-1679 vol.3, 1999.

[142] W.-Y. Choi, "A Real-Time Algorithm for MAC Throughput Enhancement by Dynamic RTS-CTS Threshold in IEEE 802-11 Wireless LANs," *Journal of RF Engineering and Telecommunications,* vol. 59, pp. 171-176, 2005.

[143] A. Rahman, and P. Gburzynski, "Hidden Problems with the Hidden Node Problem." *23rd Biennial Symposium on Communications*, pp. 270-273, 2006.

[144] W.-Y. Choi, "Clustering Algorithm for Hidden Node Problem In Infrastructure Mode IEEE 802.11 Wireless LANs." *10th International Conference on Advanced Communication Technology ICACT '08*, pp. 1335-1338, 2008.

[145] P. Sarolahti, "Congestion control on spurious TCP retransmission timeouts." *IEEE GLOBECOM '03*, pp. 682-686 vol.2, 2003.

[146] M. Changming, and L. Ka-Cheong, "Improving TCP robustness under reordering network environment." *IEEE GLOBECOM '04*, pp. 828-832 vol.2, 2004.

[147] S. Bhandarkar, N. E. Sadry, A. L. N. Reddy *et al.*, "TCP-DCR: a novel protocol for tolerating wireless channel errors," *IEEE Transactions on Mobile Computing,* vol. 4, no. 5, pp. 517-529, 2005.

[148] A. Gurtov, and R. Ludwig, "Responding to Spurious Timeouts in TCP." *IEEE INFOCOM '03*, 2003.

[149] D. Malone, D. J. Leith, A. Aggarwal *et al.*, "Spurious TCP Timeouts in 802.11 Networks." *Workshop on Wireless Network Measurement WiNMee '08*, 2008.

[150] P. Sarolahti, M. Kojo, and K. Raatikainen, *F-RTO: A New Recovery Algorithm for TCP Retransmission Timeouts*, University of Helsinki, 2002.

[151] G. Xylomenos, and G. C. Polyzos, "TCP and UDP Performance over a Wireless LAN." *IEEE INFOCOM '99*, 1999.

[152] M. Bottigliengo, C. Casetti, C. F. Chiasserini *et al.*, "Smart traffic scheduling in 802.11 WLANs with access point." *Vehicular Technology Conference VTC '03*, pp. 2227-2231 vol.4, 2003.

[153] A. C. H. Ng, D. Malone, and D. J. Leith, "Experimental Evaluation of TCP Performance and Fairness in an 802.11e Test-bed." *ACM SIGCOMM '05*, pp. 17 - 22, 2005.

[154] S. Pilosof, R. Ramjee, D. Raz *et al.*, "Understanding TCP fairness over Wireless LAN." *IEEE INFOCOM '03*, 2003.

[155] N. Blefari-Melazzi, A. Detti, A. Ordine *et al.*, "A mechanism to enforce TCP Fairness in 802.11 wireless LANs and its performance evaluation in a real test-bed." *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks WoWMoM '07*, pp. 1-7, 2007.

[156] Y. Wu, Z. Niu, and J. Zhu, "Upstream/Downstream Unfairness Issue of TCP over Wireless LANs with Per-flow Queueing." *IEEE ICC '05*, pp. 3543 - 3547, 2005.

[157] E. R. Gomez, and M. Davis, "The Impact of TCP Sliding Window on the Performance of IEEE 802.11 WLANs." *Irish Signals and Systems Conference*, pp. 231 - 234, 2006.

[158] M. Bottigliengo, C. Casetti, C. F. Chiasserini *et al.*, "Short-term Fairness for TCP Flows in 802.11b WLANs." *IEEE INFOCOM '04*, 2004.

[159] Z. Hadzi-Velkov, and B. Spasenovski, "Capture effect in IEEE 802.11 basic service area under influence of Rayleigh fading and near/far effect." *IEEE PIMRC '02*, pp. 172-176 vol.1, 2002.

[160] A. Nyandoro, L. Libman, and M. Hassan, "Service Differentiation Using the Capture Effect in 802.11 Wireless LANs," *IEEE Transactions on Wireless Communications,* vol. 6, no. 8, pp. 2961-2971, 2007.

[161] V. Kemerlis, S. Eleftherios, G. Xylomenos *et al.*, "Throughput Unfairness in TCP over WiFi." *3rd Annual Conference on Wireless On demand Network Systems and Services WONS '06*, 2006.

[162] A. Aaron, and S. Tsao, *Techniques to Improve TCP over Wireless Links*, Report EE 359, Stanford University, 2000.

[163] J. Border, M. Kojo, J. Griner *et al.*, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," *RFC 3135*, 2001.

[164] A. V. Bakre, and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts." *15th International Conference on Distributed Computing Systems ICDCS '95*, pp. 136 - 143, 1995.

[165] D. Bosau, "Path Tail Emulation: An Approach to Enable End-to-End Congestion Control for Split Connections and Performance Enhancing Proxies." *KiVS Kurzbeiträge und Workshop*, pp. 33-40, 2005.

[166] P. Sweeney, *Error Control Coding: From Theory to Practice*: John Wiley & Sons Ltd, 2002.

[167] C. Parsa, and J. J. Garcia-Luna-Aceves, "Improving TCP congestion control over Internets with heterogeneous transmission media." *Seventh International Conference on Network Protocols ICNP '99*, pp. 213-221, 1999.

[168] H. Balakrishnan, S. Seshan, E. Amir *et al.*, "Improving TCP/IP Performance over Wireless Networks," *ACM Wireless Networks,* vol. 1, no. 4, pp. 2 - 11-2 - 11, 1995.

[169] C. H. Ng, J. Chow, and L. Trajkovic, "Performance Evaluation of TCP over WLAN 802.11 with the Snoop Performance Enhancing Proxy." *OPNETWORK '02*, 2002.

[170] P. Sinha, N. Venkitaraman, R. Sivakumar *et al.*, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," *ACM Wireless Networks,* vol. 8, no. 2 - 3, pp. 301 - 316, 2002.

[171] N. H. Vaidya, M. N. Mehta, C. E. Perkins *et al.*, "Delayed Duplicate Acknowledgements: A TCP-Unaware Approach to Improve Performance of TCP over Wireless," *Wireless Communications and Mobile Computing,* vol. 2, no. 1, pp. 59 - 70, 2001.

[172] T. Goff, J. Moronski, D. S. Phatak *et al.*, "Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments." *IEEE INFOCOM '00*, pp. 1537-1545 vol.3, 2000.

[173] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: a new congestion control scheme for satellite IP networks," *IEEE/ACM Transactions on Networking,* vol. 9, no. 3, pp. 307-321, 2001.

[174] V. Tsaoussidis, and C. Zhang, "TCP-Real: receiver-oriented congestion control," *Comput. Netw.,* vol. 40, no. 4, pp. 477-497, 2002.

[175] R. Krishnan, M. Allman, C. Partridge *et al.*, "Explicit Transport Error Notification (ETEN) for Error Prone Wireless and Satellite Networks", *International Journal of Computer and Telecommunications Networking*, pp. 343 - 362, vol. 46(3), 2004.

[176] W. Xiuchao, C. Mun, and A. L. Ananda, "TCP HandOff: A Practical TCP Enhancement for Heterogeneous Mobile Environments." *IEEE ICC '07*, pp. 6043-6048, 2007.

[177] V. B. Reddy, and A. K. Sarje, "Differentiation of Wireless and Congestion Losses in TCP." *2nd International Conference on Communication Systems Software and Middleware COMSWARE '07*, pp. 1-5, 2007.

[178] S. Floyd, "A Report on Recent Developments in TCP Congestion Control." *IEEE Communications Magazine*, pp. 84-90, April 2001.

[179] S. Floyd, "Issues with TCP SACK," *Technical Report, LBL Network Group*, 1996.

[180] H. Jeng-Ji, and C. Jin-Fu, "A new method to improve the performance of TCP SACK over wireless links." *Vehicular Technology Conference VTC '03*, pp. 1730-1734 vol.3, 2003.

[181] E. Blanton, M. Allman, K. Fall *et al.*, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP," *RFC 3517*, 2003.

[182] S. Floyd, and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, 1999.

[183] J. Zhu, Z. Li, and Z. Niu, "A modified TCP-NewReno retransmission scheme for lossy network." *Fifth Asia-Pacific Conference on Communications and Fourth Optoelectronics and Communications Conference APCC/OECC '99*, pp. 204-208 vol.1, 1999.

[184] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," *IEEE/ACM Transactions on Networking,* vol. 6, no. 4, 1998.

[185] K. Soomin, C. Sunwoong, and K. Chongkwon, "Instantaneous variant of TCP NewReno," *Electronics Letters,* vol. 36, no. 19, pp. 1669-1670, 2000.

[186] S. Floyd, S. Mahdavi, M. Mathis *et al.*, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," *RFC 2883*, 2000.

[187] E. Blanton, and M. Allman, "On making TCP more robust to packet reordering," *ACM Computer Communication Review,* vol. 32, no. 1, pp. 20-30, 2002.

[188] Z. Ming, B. Karp, S. Floyd *et al.*, "RR-TCP: a reordering-robust TCP with DSACK." *11th IEEE International Conference on Network Protocols*, pp. 95-106, 2003.

[189] A. Chrungoo, V. Gupta, H. Saran *et al.*, "TCP k-SACK: a simple protocol to improve performance over lossy links." *IEEE GLOBECOM '01*, pp. 1713-1717 vol.3, 2001.

[190] K. Beomjoon, K. Dongmin, and L. Jaiyong, "Lost retransmission detection for TCP SACK," *Communications Letters, IEEE,* vol. 8, no. 9, pp. 600-602, 2004.

[191] D. Lin, and H. T. Kung, "TCP fast recovery strategies: analysis and improvements." *IEEE INFOCOM '98*, pp. 263-271, 1998.
[192] K. Beomjoon, and L. Jaiyong, "Retransmission loss recovery by duplicate acknowledgment counting," *Communications Letters, IEEE,* vol. 8, no. 1, pp. 69-71, 2004.
[193] L. Brakmo, and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal of Selected Areas in Communications (JSAC),* vol. 13, no. 8, pp. 1465-1480, 1995.
[194] Y. Leang Tzeh, J. Liew, and W. K. G. Seah, "Experimentation of TCP schemes over GPRS & WLAN." *4th International Workshop on Mobile and Wireless Communications Network*, pp. 234-238, 2002.
[195] R. Ferorelli, L. A. Grieco, S. Mascolo *et al.*, "Live Internet measurements using Westwood+ TCP congestion control." *IEEE GLOBECOM '02*, pp. 2583-2587 vol.3, 2002.
[196] L. A. Grieco, and S. Mascolo, "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control," *ACM SIGCOMM Computer Communication Review (CCR),* vol. 34, no. 2, pp. 25 - 28-25 - 28.
[197] L. A. Grieco, and S. Mascolo, "Performance evaluation of Westwood+ TCP over WLANs with local error control." *28th Annual IEEE International Conference on Local Computer Networks LCN '03*, pp. 440-448, 2003.
[198] L. Cui, J. K. Seok, P. Jung Soo *et al.*, "Enhanced Westwood+ TCP for Wireless/Heterogeneous Networks." *Asia-Pacific Conference on Communications APCC '06*, pp. 1-5, 2006.
[199] C. P. Fu, and S. C. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE Journal of Selected Areas in Communications (JSAC),* vol. 21, no. 2, 2003.
[200] Q. Pang, S. C. Liew, C. P. Fu *et al.*, "Performance Study of TCP Veno over WLAN and RED Router," *Wireless Communications and Mobile Computing,* vol. 4, no. 8, pp. 867 - 879, 2004.
[201] F. Cheng Biao, J. L. Wang, C. P. Fu *et al.*, "Performance study of TCP Veno in wireless/asymmetric links." *International Conference on Cyberworlds*, pp. 447-451, 2004.
[202] I. E. Khayat, P. Geurts, and G. Leduc, "Improving TCP in Wireless Networks with an Adaptive Machine-Learnt Classifier of Packet Loss Causes." *IFIP-TC6 Networking '05*, pp. 549 - 560, 2005.
[203] Z. Zixuan, L. Bu Sung, and F. Cheng Peng, "Packet loss and congestion state in TCP Veno." *12th IEEE International Conference on Networks ICON 2004*, pp. 731-735 vol.2, 2004.
[204] Z. Ke, and F. Cheng Peng, "The Performance Study of TCP Veno Under Different Recovery Schemes." *Asia-Pacific Conference on Communications APCC '06*, pp. 1-5, 2006.
[205] Z. ZiXuan, F. Cheng Peng, and L. Bu Sung, "A refinement to improve TCP Veno performance under bursty congestion." *IEEE GLOBECOM '05*, 2005.
[206] K. Zhang, C. P. Fu, and Z. ZiXuan, "WLC47-3: Loss Distinguishing Accuracy in TCP Veno and its Performance Influence." *IEEE GLOBECOM '06*, pp. 1-5, 2006.
[207] M. Todorovic, and N. Lopez-Benitez, "Efficiency Study of TCP Protocols in InfrastructuredWireless Networks." *International conference on Networking and Services ICNS '06*, pp. 103-103, 2006.

[208] C. Caini, and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking,* vol. 22, pp. 547-566, 2004.
[209] C. Carlo, L. Nico Candio, F. Rosario *et al.*, "TCP Hybla Performance in GEO Satellite Networks: Simulations and Testbed." *International Workshop on Satellite and Space Communications*, pp. 41-45, 2006.
[210] C. Caini, R. Firrincieli, D. Lacamera *et al.*, "TCP Live Experiments on a Real GEO Satellite Testbed." *12th IEEE Symposium on Computers and Communications ISCC '07*, pp. 523-529, 2007.
[211] E. H. K. Wu, and C. Mei-Zhen, "JTCP: jitter-based TCP for heterogeneous wireless networks," *IEEE Journal on Selected Areas in Communications,* vol. 22, no. 4, pp. 757-766, 2004.
[212] H. Schulzrinne, S. Casner, R. Frederick *et al.*, "RTP: A Transport Protocol for Real-Time Applications," *RFC 1889*, 1996.
[213] E. H. K. Wu, I. H. Ming, C. Mei-Zhen *et al.*, "JTCP: congestion distinction by the jitter-based scheme over wireless networks." *IEEE PIMRC '04*, pp. 2473-2477 vol.4, 2004.
[214] Y. Daiqin, L. Ka-Cheong, and V. O. K. Li, "Simulation-Based Comparisons of Solutions for TCP Packet Reordering in Wireless Networks." *IEEE Wireless Communications and Networking Conference WCNC '07*, pp. 3238-3243, 2007.
[215] K. K. Leung, T. E. Klein, C. F. Mooney *et al.*, "Methods to improve TCP throughput in wireless networks with high delay variability [3G network example]." *Vehicular Technology Conference VTC '04*, pp. 3015-3019 vol. 4, 2004.
[216] P. Karn, and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *SIGCOMM Computer Communication Review,* vol. 17, no. 5, pp. 2-7, 1987.
[217] R. Ludwig, and R. H. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions," *ACM Computer Communication Review,* vol. 30, no. 1, 2000.
[218] R. Ludwig, and M. Meyer, "The Eifel Detection Algorithm for TCP," *RFC 3522*, 2003.
[219] H. Ekstrom, and R. Ludwig, "The peak-hopper: a new end-to-end retransmission timer for reliable unicast transport." *IEEE INFOCOM '04*, pp. 2502-2513 vol.4, 2004.
[220] I. Psaras, V. Tsaoussidis, and L. Mamatas, "CA-RTO: a contention-adaptive retransmission timeout." *14th International Conference on Computer Communications and Networks ICCCN '05*, pp. 179-184, 2005.
[221] J. Cobb, and P. Agrawal, "Congestion or Corruption? A strategy for efficient wireless TCP sessions." *IEEE Symposium on Computers and Communications*, pp. 262-268, 1995.
[222] S. Biaz, and N. H. Vaidya, "Distinguishing congestion losses from wireless transmission losses: a negative result." *7th International Conference on Computer Communications and Networks*, pp. 722-731, 1998.
[223] D. Barman, and I. Matta, "A Bayesian Approach for TCP to Distinguish Congestion from Wireless Losses," *BUCS Technical Report 2003-030, Boston University, Department of Computer Science*, 2003.
[224] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-end differentiation of congestion and wireless losses," *IEEE/ACM Transactions on Networking,* vol. 11, no. 5, pp. 703-717, 2003.

[225] C. H. Lim, and J. W. Jang, "Robust end-to-end loss differentiation scheme for transport control protocol over wired/wireless networks," *Communications, IET,* vol. 2, no. 2, pp. 284-291, 2008.

[226] S. Bregni, D. Caratti, and F. Martignon, "Enhanced loss differentiation algorithms for use in TCP sources over heterogeneous wireless networks." *IEEE GLOBECOM '03*, pp. 666-670 vol.2, 2003.

[227] N. K. G. Samaraweera, "Non-congestion packet loss detection for TCP error recovery using wireless links," *Communications, IEEE,* vol. 146, no. 4, pp. 222-230, 1999.

[228] V. Tsaoussidis, and H. Badr, "TCP-probing: towards an error control schema with energy and throughput performance gains." *International Conference on Network Protocols*, pp. 12-21, 2000.

[229] J. Liu, I. Matta, and M. Crovella, "End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment." *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks WiOpt'03*, 2003.

[230] J. Liu, and M. Crovella, "Using loss pairs to discover network properties." *ACM SIGCOMM Internet Measurement Workshop*, 2001.

[231] S. Biaz, and N. H. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver." *IEEE Symposium on Application-Specific Systems and Software Engineering and Technology ASSET '99*, pp. 10-17, 1999.

[232] Y. Tobe, Y. Tamura, A. Molano *et al.*, "Achieving moderate fairness for UDP flows by path-status classification." *25th Annual IEEE Conference on Local Computer Networks LCN '00*, pp. 252-261, 2000.

[233] Y. Guang, W. Ren, M. Y. Sanadidi *et al.*, "TCPW with bulk repeat in next generation wireless networks." *IEEE ICC '03*, pp. 674-678 vol.1, 2003.

[234] S. Mascolo, C. Casetti, M. Gerla *et al.*, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links." *ACM MOBICOM '01*, pp. 287 - 297, 2001.

[235] M. V. Delibasic, and I. D. Radusinovic, "LD-LogWestwood+ TCP for Wireless Networks." *8th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services TELSIKS '07*, pp. 389-392, 2007.

[236] D. Kliazovich, F. Granelli, and D. Miorandi, "TCP Westwood+ Enhancement in High-Speed Long-Distance Networks." *IEEE ICC '06*, pp. 710-715, 2006.

[237] M. V. Delibasic, and I. D. Radusinovic, "Performance Evaluation of LogWestwood+ TCP in Wired/Wireless Networks." *European Wireless Conference '07*, 2007.

[238] L. Carvalho, J. Angeja, and A. Navarro, "A new packet loss model of the IEEE 802.11g wireless network for multimedia communications," *IEEE Transactions on Consumer Electronics,* vol. 51, no. 3, pp. 809-814, 2005.

[239] Y. Dae Gil, S. Soo Young, K. Wook Hyun *et al.*, "Packet Error Rate Analysis of IEEE 802.11b under IEEE 802.15.4 Interference." *Vehicular Technology Conference VTC '06*, pp. 1186-1190, 2006.

[240] M. R. Souryal, L. Klein-Berndt, L. E. Miller *et al.*, "Link assessment in an indoor 802.11 network." *IEEE Wireless Communications and Networking Conference WCNC '06*, pp. 1402-1407, 2006.

[241] W. Hneiti, and N. Ajlouni, "Performance Enhancement of Wireless Local Area Networks." *Information and Communication Technologies ICTTA '06*, pp. 2400-2404, 2006.

[242] D. Youngju, L. Seungbeom, and P. Sin-Chong, "Adaptive Acknowledgment schemes of the IEEE 802.11e EDCA." *The 9th International Conference on Advanced Communication Technology*, pp. 1679-1683, 2007.
[243] P. Chatzimisios, A. C. Boucouvalas, and V. Vitsas, "Influence of channel BER on IEEE 802.11 DCF," *Electronics Letters,* vol. 39, no. 23, pp. 1687-9, 2003.
[244] X. Chang, "Network Simulations with OPNET." *31st Winter Simulation Conference*, 1999.
[245] M. N. Akhtar, M. A. O. Barry, and H. S. Al-Raweshidy, "Modified Tahoe TCP for Wireless Networks Using OPNET Simulator." *London Communications Symposium LCS '03*, 2003.
[246] J. Song, and L. Trajkovic, "Enhancements and Performance Evaluation of Wireless Local Area Networks." *OPNETWORK '03*, 2003.
[247] M. Omueti, and L. Trajkovic, "OPNET model of TCP with adaptive delay and loss response for broadband GEO satellite networks." *OPNETWORK '07*, 2007.
[248] A. Jayananthan, H. Sirisena, and V. Garg, "Analytical Model of TCP with Enhanced Recovery Mechanism for Wireless Environments." *IEEE ICC '07*, pp. 4506-4511, 2007
[249] G. Flores-Lucio, M. Paredes-Farrera, E. Jammeh *et al.*, "OPNET-Modeler and NS-2: Comparing the Accuracy of Network Simulators for Packet-Level Analysis using a Network Testbed." *3rd WEAS Int. Conf. on Simulation, Modelling and Optimization ICOSMO '03*, pp. 700-707, 2003.
[250] "Understanding TCP Model Internals and Interfaces: Discrete Event Simulation for R&D (Session 1508)," *OPNETWORK '07*, 2007.
[251] "Understanding Wireless LAN Model Internals and Interfaces: Discrete Event Simulation for R&D (Session 1529)," *OPNETWORK '07*, 2007.
[252] E. Casilari, F. J. Gonzblez, and F. Sandoval, "Modeling of HTTP traffic," *Communications Letters, IEEE,* vol. 5, no. 6, pp. 272-274, 2001.
[253] R. Fielding, J. Gettys, J. C. Mogull *et al.*, *Hypertext transfer protocol -- HTTP/1.1*, HTTP Working Group, 1997.
[254] D. Barman, and I. Matta, "Model-based Loss Inference by TCP over Heterogeneous Networks." *Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks WiOpt '05*, 2005.
[255] A. K. Ghosh, S. Das, R. Roy *et al.*, "Sender Side Intelligence for TCP Throughput Enhancement in Wired-Cum-Wireless Network." *IEEE PIMRC '07*, pp. 1-5, 2007.
[256] M. Gerla, K. Tang, and R. Bagrodia, "TCP Performance in Wireless Multi-hop Networks." *Mobile Computing Systems and Applications*, 1999.
[257] Q. Ni, T. Turletti, and W. Fu, "Simulation-based Analysis of TCP Behavior Over Hybrid Wireless and Wired Network." *International Conference on Internet Computing*, pp. 27-36, 2002.
[258] S. Gopal, and D. Raychaudhuri, "Experimental evaluation of the TCP simultaneous-send problem in 802.11 wireless local area networks." *ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis*, pp. 23-28, 2005.
[259] C. Barakat, and E. Altman, "On ACK Filtering on a Slow Reverse Channel," *International Journal of Satellite Communications and Networking,* vol. 21, pp. 241-258, 2003.
[260] V. T. Raisinghani, A. Patil, and S. Iyer, "Mild Aggression : A New Approach for Improving TCP Performance in Asymmetric Networks." *Asian International Mobile Computing Conference*, 2000.

[261] C. Shan, B. Bensaou, and Z. Junhua, "On Estimating the Bandwidth Share of TCP Connections in Presence of Reverse Traffic." *Workshop on High Performance Switching and Routing HPSR '07*, pp. 1-6, 2007.
[262] R. Y. Awdeh, and S. Akhtar, "Comparing TCP variants in presence of reverse traffic," *Electronics Letters,* vol. 40, no. 11, pp. 706-708, 2004.
[263] "Net:Netem - Linux Foundation," www.linuxfoundation.org/en/Net:Netem.
[264] "Net:Bridge - Linux Foundation," www.linuxfoundation.org/en/Net:Bridge.
[265] J. Yu, *Performance Evaluation of Linux Bridge*, DePaul University, School of CTI, 2004.
[266] NLANR/DAST, "Iperf." http://dast.nlanr.net/Projects/Iperf/.
[267] I. McDonald, and R. Nelson, "Congestion Control Advancements in Linux." *Australia's National Linux Conference LCA '06*, 2006.
[268] X. Lisong, K. Harfoush, and R. Injong, "Binary increase congestion control (BIC) for fast long-distance networks." *IEEE INFOCOM '04*, pp. 2514-2524 vol.4, 2004.
[269] P. De, A. Raniwala, S. Sharma *et al.*, "MiNT: a miniaturized network testbed for mobile wireless research." *IEEE INFOCOM '05*, pp. 2731-2742 vol. 4, 2005.
[270] J. Liu, and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," *IEEE Journal of Selected Areas in Communications (JSAC)*, 2001.
[271] M. Kojo, A. Gurtov, and J. Manner, "Seawind: A Wireless Network Emulator." *ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, 2001.
[272] P. Zheng, and L. M. Ni, "EMPOWER: A Network Emulator for Wireline and Wireless Networks." *IEEE INFOCOM '03*, pp. 1933 - 1942, 2003.
[273] A. Wennstrom, A. Brunstrom, J. Rendon *et al.*, "A GPRS Testbed for TCP Measurements." *4th International Workshop on Mobile and Wireless Communications Network*, pp. 320 - 324, 2002.
[274] A. Hafslund, L. Landmark, P. Engelstad *et al.*, "Testing and Analyzing TCP Performance in a Wireless-Wired Mobile Ad Hoc Test Bed." *IWWAN '04*, pp. 115 - 119, 2004.
[275] M. Borri, M. Casoni, and M. L. Merani, "An experimental study on congestion control in wireless and wired networks." *IEEE ICC '05*, pp. 3570-3575 vol. 5, 2005.
[276] M. Yajnik, S. Moon, J. Kurose *et al.*, "Measurement and Modelling of the Temporal Dependence in Packet Loss." *IEEE INFOCOM '99*, pp. 345 - 352, 1999.
[277] A. Bon, C. Caini, T. D. Cola *et al.*, "An Integrated Testbed for Wireless Advanced Transport Protocols and Architectures." *Testbeds and Research Infrastructures for the Development of Networks and Communities TRIDENTCOM '06*, pp. 526 - 529, 2006.
[278] BrainSlayer. "DD-WRT," http://www.dd-wrt.com/dd-wrtv3/index.php.
[279] CACE Technologies, "AirPcap." http://www.cacetech.com/.
[280] M. Mathis, J. Heffner, and R. Reddy, "Web100: Extended TCP Instrumentation for Research, Education, and Diagnosis," *ACM SIGCOMM Computer Communication Review,* vol. 33, no. 3, pp. 69 - 79, 2003.
[281] NSF, "The Web100 Project." http://www.web100.org/
[282] S. Bassi, and M. A. Labrador, "Setting up a Web100-Dummynet Testbed for Research in Transport Layer Protocols." *ACM Southeast Regional Conference*, pp. 65 - 69, 2005.

[283] S. Floyd, "HighSpeed TCP for Large Congestion Windows," *RFC 3649*, 2003.
[284] R. N. Shorten, and D. J. Leith, "H-TCP: TCP for high-speed and long-distance networks." *2nd International Workshop on Protocols for Fast Long-Distance Networks PFLDnet '04*, 2004.
[285] T. Kelly, "Scalable TCP: Improving Performance in High Speed Wide Area Networks," *ACM Computer Communication Review,* vol. 32, no. 2, 2003.
[286] L. Xu, and I. Rhee, "CUBIC: A new TCP-Friendly High-speed TCP variant." *3rd International Workshop on Protocols for Fast Long-Distance Networks PFLDnet '05*, 2005.
[287] A. Kuzmanovic, and E. W. Knightly, "TCP-LP: Low-Priority Service via End-Point Congestion Control," *ACM Transactions on Networking,* vol. 14, no. 4, 2003.
[288] LBL, "tcpdump" http://www.tcpdump.org/.
[289] S. Ostermann, "tcptrace" http://www.tcptrace.org/
[290] "Linux Advanced Routing & Traffic Control" http://lartc.org/.
[291] B. D. Schuymer, "ebtables" http://ebtables.sourceforge.net/.
[292] B. A. Mah, "pchar" http://www.kitchenlab.org/www/bmah/Software/pchar/.
[293] G. Combs, "Wireshark" http://www.wireshark.org/.
[294] M. Milner, "Netstumbler" http://www.stumbler.net/.
[295] R. Taank, and P. Xiao-Hong, "An Experimental Testbed for Evaluating End-to-End TCP Performance Over Wired-to-Wireless Paths." *5th IEEE Consumer Communications and Networking Conference CCNC '08*, pp. 523-527, 2008.
[296] A. Grilo, and M. Nunes, "Performance evaluation of IEEE 802.11e." *IEEE PIMRC '02*, pp. 511-517 vol.1, 2002.
[297] M. J. Ho, J. Wang, K. Shelby *et al.*, "IEEE 802.11g OFDM WLAN throughput performance." *Vehicular Technology Conference VTC '03*, pp. 2252-2256 vol.4, 2003.