Adaptive Delay Injection for Improving TCP Performance in 802.11 WLANs with High Delay Variability

by

Georgios I. Fotiadis

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Masters of Science

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CRETE, HERAKLION

Fall 2005

University of Crete

Department of Computer Science

Adaptive Delay Injection for Improving TCP Performance in 802.11 WLANs with High Delay Variability

Author:

Georgios Fotiadis
Department of Computer Science

President of the Committee:

Vasilios Siris
Assistant Professor
Department of Computer Science

Members of the Committee:

Apostolos Traganitis
Professor
Department of Computer Science

Maria Papadopouli
Assistant Professor
Department of Computer Science

Approved by:

Dimitris Plexousakis
Chairman of Graduate Studies
Department of Computer Science

Heraklion, November 2005

Abstract

During the past few years, wireless local area networks (WLANs) based on the IEEE 802.11 protocol have experienced a tremendous growth. This success is primarily driven by the need for universal access to the Internet and by the ability offered to users to move freely without the need of cables. One of the challenges in wireless networks is to integrate to the existing Internet infrastructure. However, the wireless path can not guarantee reliable data transfer, introducing high packet error rates and delay variability. Hence, the interaction of existing transport protocols initially designed for wireline paths, such as TCP, with wireless protocols, such as 802.11, appears to be highly challenging. TCP performs poorly in wireless environments, mainly because packet losses are misinterpreted as congestion signals by the TCP sender. Unnecessary congestion avoidance invocations lead to transmission rate throttling by the TCP sender and, hence, reduced throughput. Moreover, local retransmissions, user mobility, and handoffs, often introduce high delay variability to the packets being transmitted. High delay variability can result in spurious timeouts, i.e. timeouts that could have been avoided if the sender's retransmission timeout (RTO) was larger. Spurious timeouts significantly degrade TCP throughput, since the congestion window is unnecessarily reduced to one segment. Moreover, they result in unnecessary segment retransmissions by the TCP sender, which consume battery power if the sending node is the mobile user.

In this work, we present an approach that improves TCP performance in both infrastructure and multihop wireless networks. Firstly, our approach focuses on preventing wireless losses by applying a robust data-link layer retransmission mechanism. Secondly, we eliminate the negative effects of high delay variability, which is caused by local retransmissions in bursty loss environments, by injecting artificial delay to TCP acknowledgements at the access point so as to indirectly influence the sender's RTO value. The proposed algorithm is implemented solely at the access point, without requiring changes at the sending and receiving nodes. Moreover, the algorithm can adapt

its behavior to network characteristics such as packet error rate, propagation delay and number of participating mobile nodes. Experiments with the NS-2 simulation tool show that the proposed adaptive delay injection method achieves significant throughput improvements and reduces unnecessary TCP retransmissions, hence reduces a mobile TCP sender's battery consumption.

Supervisor of Dissertation: Vasilios A. Siris
Assistant Professor
University of Crete

Περίληψη

Τα τελευταία χρόνια, η χρήση ασύρματων δικτύων που βασίζονται στο πρωτόκολλο 802.11 εμφανίζει σημαντική εξάπλωση. Η επιτυχία αυτή οφείλεται, κυρίως, στην ευρύτερη ανάγκη για πρόσβαση στο διαδίκτυο καθώς και στη δυνατότητα που παρέχεται στο χρήστη για ελεύθερη μετακίνηση. Μία από τις προκλήσεις στα ασύρματα δίκτυα είναι να ενσωματωθούν στην υπάρχουσα υποδομή του διαδικτύου. Παρ΄ όλα αυτά, το ασύρματο μέσο δεν παρέχει αξιοπιστία κατά τη μετάδοση, εισάγωντας μεγάλο ρυθμό λαθών και μεγάλη μεταβλητότητα στην καθυστέρηση. Έτσι, η αλληλεπίδραση υπαρχόντων πρωτοκόλλων δικτύων τα οποία έχουν σχεδιαστεί για ενσύρματα δίκτυα, όπως το TCP, με πρωτόκολλα ασύρματων δικτύων, όπως το 802.11, αποτελεί μεγάλη πρόκληση. Όμως, το πρωτόκολλο TCP παρουσιάζει μειωμένη απόδοση σε ασύρματα περιβάλλοντα, κυρίως λόγω της παρερμηνείας της απώλειας πακέτων στο ασύρματο μέσο ως σήματα συμφόρησης. Οι μη αναγκαίες κλήσεις των μηχανισμών αποφυγής συμφόρησης οδηγούν σε μείωση του ρυθμού μετάδοσης του TCP αποστολέα και, τελικά, σε μειωμένη απόδοση του πρωτοκόλλου. Επιπλέον, οι τοπικές επαναμεταδόσεις, η κινητικότητα του χρήστη και οι μεταπομπές (handoffs) εισάγουν μεγάλη μεταβλητότητα στην καθυστέρηση των πακέτων. Αυτή η μεγάλη μεταβλητότητα στην καθυστέρηση έχει ως αποτέλεσμα την μη αναγκαία λήξη (timeout) του μετρητή επαναμετάδοσης του αποστολέα, τα οποία θα μπορούσαν να είχαν αποφεχθεί αν η τιμή του μετρητή (RTO) αυτού ήταν μεγαλύτερη. Η λήξη του μετρητή επαναμετάδοσης μειώνει σημαντικά την απόδοση του TCP, αφού χωρίς να υπάρχει λόγος το παράθυρο συμφόρησης (congestion window) μειώνεται στο ένα. Επιπλέον, οδηγούν σε αχρείαστες επαναμεταδόσεις πακέτων, οι οποίες καταναλώνουν άσκοπα την ενέργεια των ασύρματων κόμβων.

Σε αυτή την εργασία, παρουσιάζουμε μία μέθοδο η οποία στοχεύει στην αύξηση της απόδοσης του πρωτοκόλλου TCP σε ασύρματα δίκτυα υποδομής και πολλαπλών αλμάτων. Η προτεινόμενη μέθοδος εξαλείφει τα λάθη στο ασύρματο μονοπάτι προσαρμόζοντας έναν ισχυρό μηχανισμό τοπικής επαναμετάδοσης. Επιπλέον, αντιμετωπίζουμε το πρόβλη-

μα της μεγάλης μεταβλητότητας στην καθυστέρηση, η οποία εισάγεται από τις τοπικές επαναμεταδόσεις σε περιβάλλοντα με εκρήξεις λαθών, προσθέτωντας τεχνητή καθυστέρηση σε πακέτα επιβεβαιώσεων στο σημείο πρόσβασης (access point). Επιπλέον, η προτεινόμενη μέθοδος προσαρμόζεται σε συγκεκριμένα χαρακτηριστικά του δικτύου, όπως ο ρυθμός λαθών, η καθυστέρηση διάδοσης και ο αριθμός των ασύρματων κόμβων. Τα πειράματα που έγιναν στον προσομοιωτή δικτύου Network Simulator 2, δείχνουν ότι ο προτεινόμενος αλγόριθμος πετυχαίνει βελτίωση στην απόδοση του TCP και μειώνει τον αριθμό των μη αναγκαίων επαναμεταδόσεων, αποφεύγοντας την άσκοπη κατανάλωση ενέργειας στους ασύρματους κόμβους.

Επόπτης Καθηγητής: Βασίλειος Α. Σύρης
Επίκουρος Καθηγητής
Πανεπιστήμιο Κρήτης

Acknowledgments

The present study is the result of personal hard work. However, the outcome of this work could have been far from success without the help of many people. Their support and willingness to help was more than invaluable and I hope that I will come up to their expectations.

Assistant professor Vassilios A. Siris, my supervisor, has a great contribution to the present work. He had always been available to discuss about my thesis and had always led me towards the right directions. This is why am I grateful to him.

I would also like to thank my friends and colleagues at the pleasant environment of ICS-FORTH. Giorgos Stamataki's friendship and priceless advices, as far as the present work is concerned, have been of great importance to me. Also, Despoina Triantafullidou was always more than available to provide her valuable ns-2 advice to me. Moreover, I would like to thank Stefanos Papadakis, Vaggelis Angelakis.

Furthermore, I would like to thank my friends for their support throughout my studies. The whole world's conference proceedings are not enough to write about the contribution of Giorgos K., Ilias, Antonis, Giorgos P., Alkis and Miltos. "The times they are a-changin" but my feelings about them will not. Last, but not least, I owe the maximum to Eleftheria for her love, support and patience during this difficult period of my life.

Nevertheless, nothing would be true if it was not for my family. My father, Giannis, my mother, Chrysoula, my sister, Maria, and my brother, Pavlos, were always there for me to support and help in any way they could. For this, I dedicate this work to them.

To my family

Στην οικογένεια μου

Contents

List of Figures **xv**

List of Tables **xvii**

1 Introduction **1**
1.1 Problem statement 2
1.2 Contribution of the present work 3
1.3 Relation to previous work 5
1.4 Thesis outline 5
1.5 Outcome 6

2 Background Theory **7**
2.1 The 802.11 wireless LAN standard 7
2.1.1 The 802.11 operation modes 8
2.1.2 The 802.11 MAC Layer 9
2.2 The Transmission Control Protocol 12
2.2.1 TCP congestion control 13
2.2.2 The TCP timestamp option 15

3 TCP performance in presence of high delay variabilities **17**
3.1 TCP over wireless networks 17
3.2 TCP spurious timeouts 18
3.2.1 Impact of spurious timeouts on TCP throughput 20
3.2.2 Impact of spurious timeouts on useful transmission ratio 21

4 ACK delay injection for improving TCP performance in bursty loss environments **25**
4.1 The delay injection method 25
4.2 Improving TCP performance in presence of bursty losses 27
4.2.1 Suppressing wireless losses 27
4.2.2 Adaptive algorithm to absorb high delay variability 28
4.3 Considering downlink traffic 32

5 Experimental Evaluation **35**
5.1 Performance metrics 36
5.2 Infrastructure networks experiments 36
5.2.1 Single user experiment 38
5.2.2 Experiments with different propagation delays 40
5.2.3 Experiments with multiple users 43
5.3 Multihop networks experiments 45
5.4 Algorithm parameter selection 48
5.4.1 Delay factor ($delay\_factor$) selection 48
5.4.2 Subtraction threshold ($subthresh$) selection 49
5.5 Comparison to other methods 50
5.5.1 Comparison to static delay injection 50
5.5.2 Comparison to TCP Westwood 51

6 Related Work **53**
6.1 Improving TCP performance over wireless links 53
6.1.1 End-to-end mechanisms 53
6.1.2 Link layer approaches 54
6.1.3 Split connections 55
6.2 Improving TCP performance in wireless networks with high delay variability 55
6.2.1 GBN retransmission policy 56
6.2.2 The Eifel algorithm 56
6.2.3 Delay jitter algorithm 57
6.2.4 Transparent TCP proxies 58
6.3 Relation to real traffic measurements 58

7 Conclusions and issues for further research **61**

Bibliography **63**

List of Figures

2.1 An Extended Service Set (ESS): Infrastructure Wireless Network 8
2.2 An Independent Basic Service Set (IBSS): Ad-hoc Wireless Network . . 9
2.3 The 802.11 CSMA/CA mechanism 10
2.4 The 802.11 RTS/CTS mechanism 11

3.1 Impact of spurious timeouts on the useful transmission ratio 22

4.1 Calculation of the *pdelay* variable 29
4.2 Delay Injection Procedure 31
4.3 The modified round robin serving mechanism 33

5.1 Infrastructure Network Topology 37
5.2 End-to-end TCP throughput and useful transmission ratio as a function of PER in a single flow experiment 38
5.3 Number of Timeouts and RTT as a function of PER in a single flow experiment 40
5.4 End-to-end TCP throughput with different wired propagation delays . . 41
5.5 Useful transmission ratio with different wired propagation delays 43
5.6 Impact of the wired propagation delay on TCP throughput for different PERs. 43
5.7 End-to-end TCP throughput and useful transmission ratio for two participating users and 15 ms wired propagation delay 44
5.8 End-to-end TCP throughput and useful transmission ratio for two participating users and 80 ms wired propagation delay 45
5.9 Aggregate throughput as a function of the number of mobile nodes for different PERs 46
5.10 Multihop network topology 46
5.11 End-to-end TCP throughput as a function of PER for two and three hops 47
5.12 Percentage difference of our method and the inf-RL as a function of the number of hops 48
5.13 Delay_factor parameter selection 49
5.14 Comparison of our method with static delay injection for different PERs 51

5.15 Comparison of DI-method to TCP Westwood 52

List of Tables

5.1 Number of timeouts as a function of propagation delay, when $PER = 20\%$ 42

Chapter 1

Introduction

Over the last decade, wireless communication systems have experienced a detonating growth. Wireless connectivity allows users to move freely while connected to a wireless local area network (WLAN), while, at the same time, it renders such a network set up an easy task, since it does not require cables or infrastructure. This development is primarily driven by the transformation of what has been largely a medium for supporting mobile telephony into a medium for supporting other services, such as the transmission of video, images, text, and data.

In 1997, the Institute of Electronic and Electrical Engineering (IEEE) released the 802.11 as the first international standard for wireless local area networks, specifying two data rates of 1 and 2 Mb/s. Two years later became available the 802.11b and 802.11a, operating at a maximum data rate of 11 and 54 Mb/s, respectively. Furthermore, in June 2003, a third standard was ratified, 802.11g, operating at a maximum data rate of 108 Mb/s. Later in June 2005, the 802.11e standard was introduced defining a set of Quality of Service enhancements for LAN applications. The research on wireless communications is on continuous progress, targeting at maximum speeds of 540 Mb/s (802.11n).

One of the challenges in wireless communication systems is their integration into today's Internet infrastructure, providing their clients with the ability to access existing

information as well as with the experience of world wide connectivity. Though, in order for this integration to be proper, existing network transport protocol, such as the Transmission Control Protocol (TCP), must perform efficiently on top of emerging wireless protocols. Nowadays, 90% of the total Internet traffic is TCP traffic [1, 2], so it is very important for this protocol to interact properly with the most popular wireless protocol; i.e. 802.11.

1.1 Problem statement

TCP was originally designed for wireline paths; hence, the particular characteristics of wireless paths impose anomalies in its proper operation. High and bursty packet error rates (PERs), introduced by the lossy wireless channel, and high delay variability, introduced by data-link layer retransmissions, mobility and handoffs, reduce significantly TCP's performance and render flow and congestion control in such environments a difficult task.

TCP provides mechanisms so that the sender controls the amount of data injected to the path, in order to avoid network congestion. Hence, the TCP sender probes the available bandwidth of the network path between the source and destination nodes and adapts its transmission rate accordingly. In wireline networks, network congestion is detected when the sender realizes that some transmitted packet(s) is (are) lost. The TCP sender detects packet losses with the aid of the following two methods: (i) expiration of its retransmission timer, and, (ii) reception of three duplicate acknowledgements. The TCP sender assumes that packet losses are attributed to buffer overflow at some intermediate router's queue and, therefore, throttles its transmission rate by reducing its congestion window (*cwnd*).

However, network congestion is not always the reason for packet losses in wireless environments. Error-prone transmissions over the radio path and high delay variability, which can lead to timeouts, are misinterpreted as signals of congestion by the TCP sender that unnecessarily reduces its transmission rate. Robust local error recovery

mechanisms mitigate the problem of wireless losses but impose additional delay to packets being locally retransmitted. Due to the burstiness of wireless channel errors, local error recovery mechanisms result in highly variable packet transmission delays, which can have a serious impact on TCP performance.

In this paper, we focus on the impact of high delay variabilities, often referred to as delay spikes, on TCP performance. Delay spikes mostly appear in environments with bursty losses, where multiple consecutive data-link layer frames can be corrupted. Delay spikes can be attributed to data-link layer retransmissions, user mobility and handoffs. Thus, they result in a sudden delay increase of specific TCP segments in contrast to the delay experienced by other segments of the same TCP session. This sudden increase to the delay of a segment, and therefore of its round trip time, can lead to spurious TCP timeouts, i.e., timeouts that could have been avoided if the retransmission timer (RTO) value of the sender was larger. Spurious timeouts have a negative impact on TCP throughput, since the congestion window ($cwnd$) is unnecessarily reduced to one segment. This negative impact becomes even more serious in emerging high bandwidth wireless networks, where the bandwidth delay product (BDP) is fairly large and, as a result, the $cwnd$ can obtain large values. Moreover, the lossy and high-delay wireless environment makes it difficult for the TCP sender to quickly recover the optimal $cwnd$ value. Finally, spurious timeouts result in unnecessary segment retransmissions by the TCP sender, since packets are not actually lost, but only delayed. These unnecessary retransmissions, which can be plenty in high bandwidth-delay environments, needlessly consume additional network resources, such as energy from the mobile nodes' battery.

1.2 Contribution of the present work

In this work, we present an approach that improves TCP performance in both infrastructure and multihop wireless environments. The performance improvements concern the end-to-end TCP throughput and the useful transmission ratio; the latter is defined as the ratio of the useful bytes transmitted to the total amount of bytes

transmitted, at transport layer. The total amount of bytes includes packets that were unnecessarily retransmitted by the TCP sender. A smaller value of the useful transmission ratio for a specific amount of data indicates that a larger amount of packets and, hence, of battery power, is required in order to transmit the data volume. Our approach focuses on eliminating wireless losses by applying a robust link layer retransmission mechanism. Persistent local retransmissions prevent packet losses even in a bursty loss environment and, as a result, solve the problem of packet losses being misinterpreted as congestion signals. Furthermore, we address the negative effects of high delay variability, which are caused by retransmission in bursty loss environments, by injecting artificial delay to specific acknowledgement (ACK) packets at the access point (AP) to indirectly influence the retransmission timeout (RTO) value of the TCP sender. Hence, the delay injection method intends to render the TCP sender's retransmission timer more robust against delay variabilities and, therefore, prevent spurious timeouts.

The main advantage of our method is that it does not require changes to the existing TCP implementation at the sending or receiving nodes. The proposed delay injection algorithm can be implemented solely at the access point. Moreover, the proposed algorithm adapts according to specific network characteristics, such as propagation delays, number of participating mobile nodes, and packet error rates (PERs) at the wireless radio path. Hence, the proposed method can be applied to a wide variety of network topologies.

Our method was tested using the Network Simulator 2 (ns-2). Using ns-2, we showed the improvements achieved on TCP performance in wireless environments with bursty losses and how these improvements depend on the propagation delay of the path, number of participating wireless stations, and PER. We tested the proposed method for both infrastructure wireless LANs and multihop wireless networks connected to wireline backbone.

1.3 Relation to previous work

The work most related to ours is [3], that shows that the injection of static or random delays at static or random intervals can lead to throughput improvements in GPRS networks. However, because of the static values of the delay injected, the performance of this approach is highly dependent on specific network characteristics, such as propagation delays, number of mobile nodes, and bit error rates. Furthermore the method is implemented at the TCP sender, whereas our method is implemented at the AP, without requiring any modification to the end systems.

Other methods [4, 5] focus on changing the retransmission timer calculation formula to make the retransmission timer more robust to large changes of the round trip time (RTT) of a TCP session. Recall that RTT is time needed for a TCP segment to reach the destination and the correspoinding ACK to travel back to the sender. All such approaches require changes at the TCP senders. Moreover, such changes may improve TCP throughput in presence of high delay variability but introduce uncertainties in the operation of TCP in wireline environments, since making the retransmission timer more robust by changing its calculation formula may delay the detection of congestion in wired environments.

Finally in [6, 7] the authors propose an algorithm, implemented at the TCP sender, so as to distinguish between packet losses due to wireless transmission and due to congestion. As with the approaches mentioned before, this algorithm is implemented at the TCP sender. Moreover, the first unnecessary retransmission can not be avoided, imposing overhead to the algorithm's performance.

1.4 Thesis outline

The rest of this report is organized as follows:

In Chapter 2, we present the basic features of wireless LANs and the most popular wireless architectures. Furthermore, we introduce the fundamentals of the two proto-

cols, whose interaction is studied in this work; the wireless 802.11 protocol and the Transmission Control Protocol (TCP).

In Chapter 3, we introduce performance-critical issues of TCP's operation in wireless environments. More specifically, we analyze TCP performance in environments with high delay variability and show their negative effect on the end-to-end TCP throughput and the useful transmission ratio.

Further, in Chapter 4, we describe our method for improving TCP performance, that can be applied to both infrastructure and multihop wireless LANs.

In Chapter 5, we present and discuss the numerical results of the application of the proposed method in wireless LANs using the Network Simulator 2 (NS-2) simulation tool.

Later, in Chapter 6, we give a brief summary of related work as far as TCP performance improvements in wireless networks is concerned, identifying where it differs from our work.

Finally, in Chapter 7, we conclude our work by summarizing the main findings and give some future research directions.

1.5 Outcome

Part of the the work presented in this report has been included in the proceedings of the International Symposium in Wireless Communication Systems (ISWCS '05), held in Sienna, Italy in September 2005, with the title "Improving TCP Throughput in 802.11 WLANs with High Delay Variability", by Georgios I. Fotiadis and Vasilios A. Siris.

Chapter 2

Background Theory

In this chapter, we briefly summarize two protocols used in this work; the 802.11 wireless protocol and the Transmission Control Protocol.

2.1 The 802.11 wireless LAN standard

The 802.11 standard addresses wireless local area networking where the connected devices communicate over the air to other devices that are within close proximity to each other, by defining the medium access control (MAC) and physical (PHY) layers.

The 802.11 standard is similar in most respects to the IEEE 802.3 Ethernet standard. Specifically, it addresses the following issues:

- Functions required for 802.11 compliant devices to operate either in a peer-to-peer fashion or integrated with an existing wired LAN
- Operation of an 802.11 device with possibly overlapping 802.11 wireless LANs and concerning the mobility of this device within multiple wireless LANs
- MAC level access control and data delivery services to the upper layers of the 802.11 network
- Physical layer signaling techniques and interfaces

- Privacy and security of user data being transferred over the wireless medium

The 802.11 protocol defines two pieces of equipment; the *wireless station*, which is usually a PC equipped with a wireless network interface card, and the *access point (AP)*, which acts as a bridge between the wireless and wired networks. The access point usually consists of both a wireless and a wired network interface as well as bridging software conforming to the 802.1d standard.

2.1.1 The 802.11 operation modes

The 802.11 standard defines two operation modes: the *infrastructure* mode and *ad hoc* mode. In *infrastructure* mode, the wireless network consists of at least one access point, connected to a wireline network, and a set of wireless stations. This simple configuration is called a Basic Service Set (BSS). All stations in the BSS communicate via the access point and not directly to each other. The Extended Service Set (ESS), extends the range of mobility to an arbitrary range, by combining two or more BSSs, forming a single subnetwork. The means by which an access point of a BSS communicates with another access point of another BSS is called Distribution System (DS); e.g. an Ethernet backbone infrastructure. An 802.11 ESS is depicted in Figure 2.1.

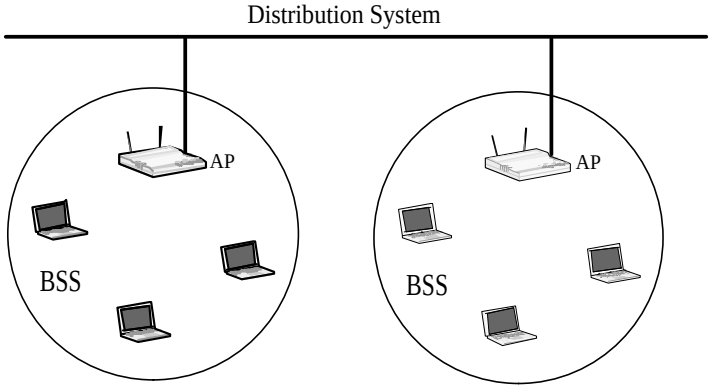


Figure 2.1: An Extended Service Set (ESS): Infrastructure Wireless Network

Ad hoc mode, also called Independent Basic Service Set (IBSS), is a set of 802.11 wireless stations that communicate directly with one another without an access point

infrastructure. An *adhoc* network is shown in Figure 2.2.

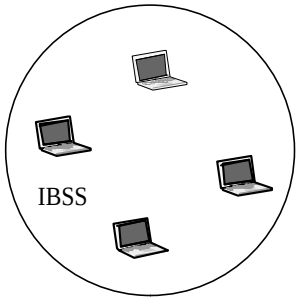


Figure 2.2: An Independent Basic Service Set (IBSS): Ad-hoc Wireless Network

The *ad hoc* mode is useful for quickly and easily setting up a wireless network anywhere that a wireless infrastructure does not exist or is not required. One common use is to create a short-lived network to support a meeting in a conference room.

2.1.2 The 802.11 MAC Layer

The 802.11 MAC layer provides a variety of functions that manage and maintain communication between 802.11 wireless stations by coordinating access to a shared radio channel. Hence, it provides functionality to allow reliable data delivery over the wireless physical medium. The data delivery itself is based on an asynchronous, best-effort, connectionless delivery of MAC layer data. There is no guarantee that the frames will be delivered successfully. Other basic functions of the 802.11 MAC layer are: data transaction, scanning, authentication, encryption, RTS/CTS and fragmentation.

The MAC 802.11 basic access method

The access to the shared wireless medium is controlled by coordination functions. The basic access method, Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA), is provided by the distributed coordination function (DCF). If contention-free service is required, it can be provided by the point coordination function (PCF), which is built on top of the DCF. To gain priority over standard contention-based services, the PCF allows stations to transmit frames after shorter intervals than the DCF.

CSMA/CA works by a "listen before talk" scheme. This means that a station wishing to transmit must first sense the radio channel to determine if another station is transmitting. If the medium is not busy, the transmission may proceed. The CSMA/CA protocol avoids collisions among stations sharing the medium by utilizing a random backoff time if the station's physical or logical sensing mechanism indicates a busy medium. During periods in which the channel is idle, the node wishing to transmit decreases its backoff counter. When the backoff counter reaches zero the node transmits the frame. This type of multiple access ensures fair channel sharing while avoiding collisions. The 802.11 CSMA/CA mechanism is depicted in Figure 2.3

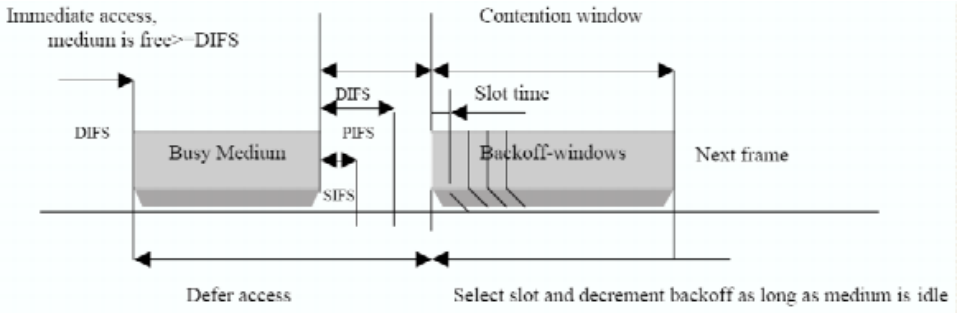


Figure 2.3: The 802.11 CSMA/CA mechanism

CSMA/CA uses three different types of interframe spaces to coordinate priorities in accessing the medium. The SIFS is used for the highest priority transmissions such as acknowledgements and RTS/CTS frames. The PIFS is used by the PCF during contention free operation. Finally, the DIFS is the minimum medium idle time for contention-based services.

The MAC 802.11 RTS/CTS mechanism

In order to minimize packet collisions, steming from the fact that a station may not be able to hear another transmitting node, the 802.11 MAC layer has a virtual carrier sensing mechanism, the RTS/CTS. According to this mechanism, a station wishing to transmit data firstly sends a control frame, the Request-to-Send (RTS) frame,

which includes the source, destination address and the duration of the transaction. The duration of the transaction is stored in the *duration* field. If the medium if free, the destination node responds with the Clear-to-Send (CTS) frame and, afterwards, the transaction can begin. Stations receiving either an RTS or a CTS, set their virtual carrier sense indicator, called network allocation vector (NAV), for the given duration and wait for this specific amount of time before they sense the medium again. By using NAV, stations can ensure that data transaction, after the RTS/CTS handshake, will not be interrupted. The operation of the RTS/CTS mechanism is depicted in Figure 2.4.

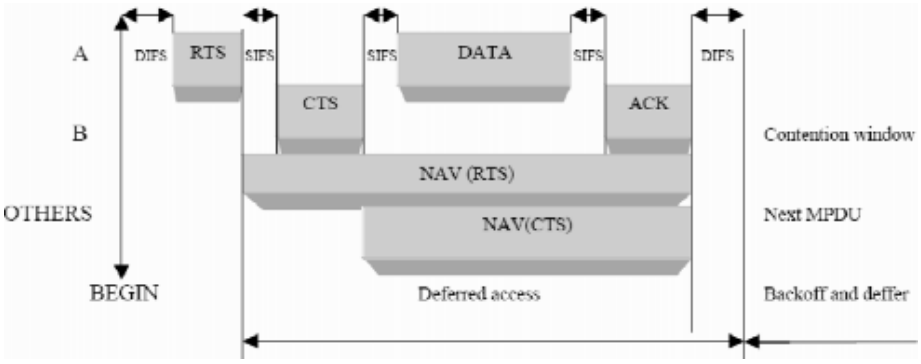


Figure 2.4: The 802.11 RTS/CTS mechanism

The RTS/CTS mechanism is governed by the MAC 802.11 $RTSThreshold$; the RTS/CTS handshake is used only for frames that are longer than the $RTSThreshold$. The default value of the $RTSThreshold$ parameter, as defined in the 802.11 standard, is 2348 bytes.

The MAC 802.11 retransmission mechanism

Like most other network protocols, 802.11 provides reliability through retransmission. Successful data transmission comprise from data frame delivery and reception of the corresponding acknowledgement frame. Hence, the entire frame transaction must complete for a transmission to be successful. When a station transmits a frame, it must receive an acknowledgment from the receiver within a specific time interval or it will

consider the transmission to have failed.

Failed transmissions increment a retry counter associated with every frame (or fragment). All mobile stations keep two retry counters: the *short retry count* and the *long retry count*. Frames that are shorter than the $RTSThreshold$ are associated with the *short retry* counter, while frames longer that the $RTSThreshold$ are associated with the *short retry* one. Frames assigned to the *short retry* counter are transmitted a maximum $ShortRetryLimit$ number of times, while for the ones associated with the *long retry* counter, $LongRetryLimit$ number of times. If the retry counter for a specific frame reaches the corresponding maximum threshold value, the frame is dropped and the transmission failure is reported to the upper layers. The default values, as the 802.11 standard defines, for the $ShortRetryLimit$ and $LongRetryLimit$ are 7 and 4, respectively.

Applying large values to the $RetryLimit$ parameters denotes a robust data link retransmission mechanism, since frames are transmitted a lot of times before being dropped. Hence, the problem of frame corruption is treated locally, at the mobile nodes or the access point. On the other hand, assigning small $RetryLimit$ values suggests handling successful packet delivery in an end-to-end fashion; hence, reliable data transaction is considered by the transport layer protocol.

2.2 The Transmission Control Protocol

The Transmission Control Protocol (TCP) is a connection-oriented, end-to-end reliable protocol build on top of the Internet Protocol (IP). TCP makes up for IP's deficiencies by providing reliable, stream-oriented connections. Hence, TCP guarantees that the data sent by the TCP sender will reach their destination uncorrupted and in the order they were sent. This reliability is build by adding sequencing information and some kind of checksum to each packet sent. Also, TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received. Besides reliable packet delivery, TCP provides congestion control, i.e. the

means for detecting that the network becomes overloaded and the actions to be followed in order to alleviate the congestion problem. We will focus on TCP congestion control by presenting, later in this section, the most important functions and algorithms needed to provide such a functionality.

2.2.1 TCP congestion control

TCP congestion control is implemented with the aid of two variables that are added per TCP connection; the congestion window and the receiver window. The congestion window (cwnd) is a sender-side limit on the amount of data that the sender can transmit into the network before receiving an acknowledgment (ACK), while the receiver's advertised window (rwnd) is a receiver-side limit on the amount of data that the receiver can accept. The minimum of cwnd and rwnd governs data transmission. TCP detects congestion by the following two methods: expiration of the sender's retransmission timer and reception of a specific number (usually three) of duplicate acknowledgements.

TCP maintains a timer for every segment's transmission; the retransmission timer. When the sender transmits a TCP segment, the retransmission timer is started. If the sender does not receive an acknowledgement for the corresponding segment before the timer expires, then the segment is regarded lost and, hence, has to be retransmitted. Then, the segment is retransmitted and the retransmission timer is re-started. The determination of the retransmission timer value (RTO) for a specific segment is based on the round trip time (RTT) of the successfully transmitted preceding segments. Hence, for the $k-th$ incoming TCP acknowledgement, the RTO is defined by the following equations:

$$SRTT[k] = (1-a) * SRTT[k-1] + a * RTT[k] \tag{2.1}$$

$$VRTT[k] = (1-b) * VRTT[k-1] + b * (|RTT[k] - SRTT[k]|) \tag{2.2}$$

$$RTO = SRTT[k] + c * VRTT[k] \tag{2.3}$$

This formula calculates RTO as the sum of the exponential smoothing average of RTT ($SRTT$) plus the its variance ($VRTT$). Typical values for a, b and c, as defined in [8], are $\frac{1}{8}$, $\frac{1}{4}$ and 4 respectively.

The second way for the TCP sender to detect congestion is through the reception of duplicate acknowledgements, i.e. ACKs that acknowledge other than the last segment sent. A reception of, usually, three duplicate ACKs with the same sequence number, lead the TCP sender conclude that the segment with sequence number at the duplicate ACKs never reached the destination and is, therefore, considered lost. Hence, the corresponding packet is retransmitted without having to wait for the retransmission timer to expire.

After TCP has detected congestion using one of the two aforementioned methods, the sender follows specific methods to alleviate the congestion problem. TCP congestion control implements the following four algorithms:

- Slow start: According to the slow start algorithm, the TCP sender increments its *cwnd* by one for every incoming ACK that acknowledges data. This results in *cwnd* doubling every RTT and, hence, suggests an exponential growth of the *cwnd*. Slow start ends when cwnd exceeds $ssthresh$ or when congestion is observed. Parameter $ssthresh$, is typically set to $\frac{cwnd}{2}$ when the retransmission timer of the TCP expires. The slow start algorithm is followed at the beginning of a new connection (to probe the available bandwidth of the path) and when the retransmission timer of the TCP sender expires, denoting heavy congestion at the network.

- Congestion avoidance: During congestion avoidance, the cwnd is incremented by one full-sized segment per round-trip time (RTT), suggesting linear growth of the *cwnd*. Congestion avoidance is followed when, during slow start phase, the *cwnd* value exceeds $ssthresh$ and continues until congestion is detected. It is also followed when light congestion, though reception of three duplicate acknowledgements, is detected.

- Fast retransmit: According to the fast retransmit algorithm, TCP retransmits a segment when a specific number (usually three) of duplicate ACKs arrive at the sender. Thereafter, TCP halves the *cwnd* and enters the congestion avoidance phase.

- Fact recovery: After the fast retransmit algorithm sends what appears to be the missing segment, the "fast recovery" algorithm governs the transmission of new data until a non-duplicate ACK arrives. According to the fast recovery algorithm, TCP increases the *cwnd* by one for every duplicate ACK it receives.

2.2.2 The TCP timestamp option

In most TCP implementations, the RTO calculation formula, comprised from Eqs. (2.1), (2.2) and (2.3), is based upon a sample of only one segment per congestion window. While this yields in an adequate approximation of the RTT for small *cwnd*s, it results in poor RTT estimates in environments with large bandwidth-delay products, such as emerging wireless networks. As a result, more accurate RTT measurements are required.

The TCP timestamp option [9] is an option provided by the TCP protocol to solve the problem of inaccurate RTT measurements. If the TCP timestamp option is enabled, then the sender the sender places a timestamp in each data segment transmitted, indicating the time that the corresponding segment was sent by the sender, and the receiver reflects the timestamp back in ACK segments. The timestamp option adds an overhead of 10 bytes to every transmitted TCP segment. However, this 10-byte overhead corresponds to, only, about 1% of a conventional TCP segment length (usually being 1400 bytes) and, hence, can be regarded negligible.

Chapter 3

TCP performance in presence of high delay variabilities

3.1 TCP over wireless networks

TCP was initially designed for wired networks but the growing popularity of mobile applications has established its deployment in wireless environments as well. TCP performs well over wireline networks by adapting the volume of data that the sender transmits according to end-to-end delays and congestion losses. However, wireless networks pose different path characteristics, which can lead to severe TCP performance degradation.

According to the TCP operation, packet losses are the most convenient way to detect congestion in wireline networks where physical layer related errors are very rare and most losses are due to buffer overflow at the intermediate router queues. As described in Chapter 2, when a packet loss is detected the sender reduces its transmission rate, so to allow the overloaded buffers to drain. However, network congestion is usually not the reason for packet losses in wireless networks. Error-prone transmissions over the wireless channel are misinterpreted as signals of congestion from the TCP sender, which unnecessarily throttles the transmission rate by reducing the congestion window (*cwnd*)

and by invoking congestion avoidance routines. Therefore, the deflated *cwnd* limits the data that are allowed to be injected to the network, i.e. the network carries less data that it can actually handle.

The performance of TCP can further degrade when multiple TCP segments, belonging to the same window, are dropped at the wireless radio channel. In this case, the impact of packet losses being misinterpreted as signals of congestion has an even more negative effect on TCP performance. TCP is shown to recover very slowly from such a situation; the authors in [10] show that TCP recovers by one lost segment per round trip time. This multiple packet loss scenario is not far from reality in bursty loss environments, where the wireless channel remains in a *bad state* for a significant amount of time. It is assumed that during the *bad state* hardly any packet can be successfully transmitted over the radio path.

3.2 TCP spurious timeouts

Besides packet losses being misinterpreted as congestion signals, TCP comes up with another unpleasant phenomenon in mobile environments. The wireless radio channel often introduces high delay variability to packets transmitted over it, leading to unpleasant effects as far as TCP performance is concerned. High delay variability can be attributed to one of the following:

- Link-layer retransmissions: In bursty loss environments packets being transmitted over the wireless medium can be corrupted at the data-link layer in a consecutive way. As a result, the mobile nodes (or the access point) spend a respectful amount of time on retransmitting the packets that are continuously corrupted. These local retransmissions introduce high delay to packets that undergo retransmission in comparison to other packets that were transmitted without any problem.

- User mobility: Mobile users often move inside their basic service set (BSS) area. The fluctuating distance between the mobile user and the access point results

in different transmission times in packets of the same TCP session and, hence, introduce delay variability.

- Handoffs: Extensive user movement in a wireless environments can often lead to users changing cells and, therefore, the network automatically switches coverage responsibility from one base station to another. The switching process is called handoff and requires a significant amount of time to accomplish. This additional amount of time needed is experienced by the mobile user as high delay variability.

In this work, we focus on delay variability that is introduced by bursty packet losses at the data-link layer. In order to analyze the operation of TCP in a bursty loss environment, we consider a mixed wired and wireless LAN topology, where all packets exchanged between the sender and the receiver cross the access point of the WLAN. Suppose that until time t, the wireless channel is in a *good state* and, therefore, successful packet transaction takes place. Let RTO_{ut} be the value of the sender's retransmission timer as it is determined by the successfully received acknowledgement packets, until time t. Now, at time t the channel enters a *bad state*, where hardly any packet can be transmitted over the radio path. Packets that were set for transmission by the mobile nodes are corrupted and, thus, have to be retransmitted. Supposing that the wireless channel remain in the *bad state* for a large amount of time, a corrupted packet can be retransmitted several times, leading to substantial increment of its RTT. This sudden delay increment, often referred to as *delay spike*, of the RTT of a specific TCP segment can result in the expiration of the retransmission timer of the TCP sender; this will happen if the RTT of the segment that undergoes local retransmission exceeds RTO_{ut}.

This timeout that the TCP sender experiences is called *spurious*, since it could have been avoided if the retransmission timeout (RTO) value of the TCP sender was larger. Upon a spurious timeout, the sender reduces its *cwnd* to one segment, leading to severe throughput degradation. Moreover, the TCP segment that triggered the spurious timeout, along with all the other segments of the same window, are unnecessarily retransmitted. These unnecessary retransmissions lead, also, to substantial reduction of the

useful transmission ratio, which results in wasteful consumption of network resources, such as the battery energy of the wireless nodes. The impact of spurious timeouts on both the TCP throughput and the useful transmission ratio is shown in the following two sections.

3.2.1 Impact of spurious timeouts on TCP throughput

As explained before, high delay variability can lead to expiration of the sender's retransmission timer and, hence, to unnecessary *cwnd* reduction to one segment. After the *cwnd* deflation, the TCP sender enters the slow start phase and starts increasing the *cwnd* in an exponential way; the phase until the sender fully recovers the value of *cwnd* just before the timer, unnecessarily, expired is called *congestion window restoration phase*. This period of time reflects the time that the network path is under-utilized, since the TCP sender transmits with a rate lower than the network can accept.

In order to estimate the impact of a spurious timeout on TCP throughput, suppose that RTT is the mean round trip time of a TCP session and that a is the value of the *cwnd* when a spurious timeout at the TCP sender occurs. We assume that a spurious timeout takes place after the *cwnd* has stabilized in its optimum value, i.e. the value of *cwnd* at which TCP operates after the initial startup phase. Recall that, the optimum value of the *cwnd* for a TCP session is equal to the bandwidth-delay product (BDP) of the path, which denotes the maximum amount of data needed to fill the path. In other words, the BDP gives an upper bound of the *cwnd*, since the path can not carry more than BDP data. Hence,

$$a \approx bandwidth * delay \tag{3.1}$$

From Eq. 3.1, we clearly see the dependence of the *cwnd restoration phase* on the *bandwidth* and the *delay* (hence the RTT) of the path. Note that, the *bandwidth* of a path is the effective rate at which a single flow can transmit.

Also, after the spurious timeout, the TCP sender enters the slow start, where

the *cwnd* is incremented in an exponential way. The increment of the *cwnd* is strongly dependent on the delay of the path, since during slow start phase the *cwnd* is doubled every RTT. Hence, the larger the delay of the path is, the longer the *cwnd restoration phase* lasts. Hence, the network remains under-utilized for a larger amount of time, decreasing end-to-end throughput severely. As a result, the impact of spurious timeouts on TCP throughput is stronger in network paths with large bandwidth and RTTs.

3.2.2 Impact of spurious timeouts on useful transmission ratio

Besides end-to-throughput, spurious timeouts also have a negative impact on the useful transmission ratio, due to unnecessary retransmissions of TCP segments by the sender. The impact of spurious timeouts on the useful transmission ratio is depicted in Fig. 3.1, which shows the packet and acknowledgment transmissions between the sender and the receiver of a TCP session. The vertical axis of Fig. 3.1 depicts the time. The Figure also shows the contents of the sending node's wireless card buffer. Also, there are two potential states that the wireless radio channel enters to: the *good state*, where packet transmission takes place without any problem, and the *bad state*, where hardly any packet can be successfully transmitted over the radio path.

So, at time t_1 packet p_1 is set for transmission by the mobile node, which also has packets p_2, p_3 and p_4 buffered at its wireless card. At time t_2, the radio channel enters a *bad state*, leading to p_1 unsuccessful transmission. Though, due to the local error recovery mechanism, packet p_1 is retransmitted by the mobile node. However, the channel remains in *bad state* for a significant amount of time, yielding to more than one local retransmissions of p_1[1]. Continuous local retransmission of p_1 introduces high delay variability to the corresponding TCP segment and, as a result, at time t_3 the retransmission timer of the TCP sender expires; a spurious timeouts occurs. So, at time t_4, packet p_1 is retransmitted by the TCP sender and is, therefore, put at the buffer of the wireless card, right after packets p_2, p_3 and p_4.

[1]Recall that the mobile nodes implement a strong local error recovery mechanism that suggests many retransmissions of a corrupted packet.

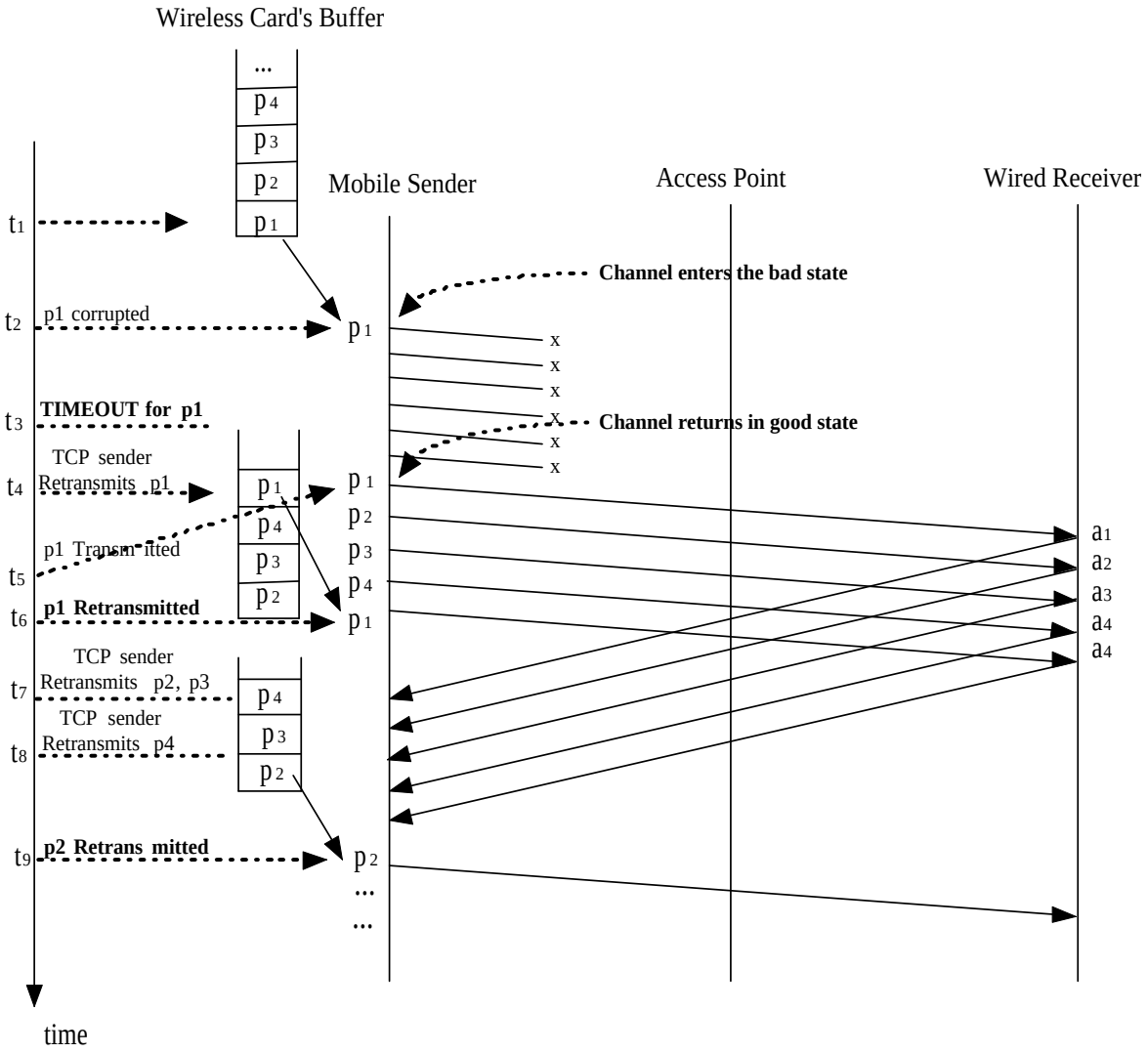


Figure 3.1: Impact of spurious timeouts on the useful transmission ratio

Now, at time t_5, the wireless channel returns in *good state* and, hence, packet p_1 is successfully transmitted over the radio path. After p_1, all other packets that lie at the wireless card's buffer (p_2, p_3, p_4 and the duplicate of p_1) are successfully transmitted. Hence, at time t_6, packet p_1 is unnecessarily transmitted by the mobile node. At time t_7, the acknowledgement for the first transmission of packet p_1 travels back to the TCP sender. The sender misinterprets acknowledgement a_1 to be for the retransmitted packet

p_1 (the one transmitted after the timeout) and unnecessarily retransmits packets p_2 and p_3, while being in slow start phase. Packets p_2 and p_3 are put at the wireless card's buffer and are later retransmitted by the mobile node. This process continues until all packets that were on the card's buffer when the spurious timeout occurred, are retransmitted.

As explained in Fig. 3.1, we see that the negative impact of spurious timeouts the useful transmission ratio is even bigger for high *cwnd*s. As a result, this impact is dependent on the bandwidth-delay product of path, as with TCP throughput. Hence, the negative effects of spurious timeouts on the useful transmission ration are even more significant in high bandwidth and delay (RTT) paths.

Chapter 4

ACK delay injection for improving TCP performance in bursty loss environments

In the previous Chapter, we showed the negative impact of spurious timeouts on the TCP end-to-end throughput and the useful transmission ratio. The impact is greater on high bandwidth-delay paths, which will be the case for emerging and future wireless communication systems. Indeed, the bandwidth of wireless networks is steadily increasing and, furthermore, there is an increasing use of wireless metropolitan area networks for Internet access. In this Chapter, we present our methodology for avoiding spurious timeouts, and, hence, improving TCP performance, in wireless networks with Internet connectivity.

4.1 The delay injection method

As stated before, spurious timeouts occur because instantaneous delay variability is misinterpreted by the TCP sender as signal of congestion, leading to unnecessary expiration of its retransmission timer. Hence, a spurious timeout could have been avoided

if the sender's retransmission timer was aware of the sudden delay increment and would not expire. So, an effective solution, without having to change the existing implementation of the TCP protocol or use an explicit signaling protocol, is to indirectly influence the value of the TCP sender's retransmission timer. As presented in Chapter 1 the retransmission timer of the TCP sender is strongly dependent on the variance of the RTT. Hence, an effective way for making the sender's retransmission timer more robust against delay variabilities is to, indirectly, increase the variance of the RTT by injecting artificial delay to specifically chosen packets.

The delay injection method has two basic parameters that need to be defined; the magnitude and frequency of the delay to be injected. The magnitude determines how much delay will be injected, while the frequency determines which packets will be delayed. So, by artificially delaying some packets of a TCP session, the variance of the RTT can be increased. Thus, the retransmission timeout value (RTO) would receive larger values, according to Eqs. (2.1), (2.2) and (2.3), and is, therefore, rendered more robust again high delay variabilities.

However, the delay injection method introduces an interesting tradeoff; artificial delay injection makes the sender's retransmission timer more robust against delay variabilities but increases the average RTT of the session. Because end-to-end-throughput is inversely proportional to the RTT, an increase of the RTT can decrease the end-to-end throughput.

In [3] the authors use the delay injection method to improve throughput in a GPRS network. The proposed method suggests injecting static or random delay at static or random intervals to TCP segments at the sender. Indeed, the method achieves throughput improvements by preventing spurious timeouts. Though, the optimal values of the magnitude and frequency of the injected delay differ according to the wireless environment characteristics and, hence, the static parameter selection can not adapt to different environments. The method we propose tries to eliminate this shortcoming by introducing an adaptive delay injection algorithm.

4.2 Improving TCP performance in presence of bursty losses

In this section, we present our methodology for improving TCP performance in wireless LANs with bursty packet errors. We, mainly, focus on uplink traffic, i.e. data traffic flowing from the mobile senders to the fixed receivers and acknowledgements to the opposite direction. Our method solves the problem of reduced throughput using two basic ideas: (a) eliminating bursty wireless losses using a robust local error recovery mechanism at the data-link layer, (b) absorbing high delay variabilities, due to local retransmissions, by injecting artificial delay to acknowledgement (ACK) packets at the access point.

4.2.1 Suppressing wireless losses

As discussed in Chapter 3, TCP in wireless environments suffers from the misinterpretation of packet losses at the radio path as congestion signals. We avoid this unpleasant phenomenon by adapting a robust retransmission mechanism at data link layer. As stated in Chapter 2, two parameters govern the retransmission mechanism of the 802.11 protocol: the *ShortRetryLimit* and *LongRetryLimit*, which denote the maximum number of retransmission for *short* and *long* frames, respectively. We use the same number of retransmissions for all data-link layer frames, regardless of their size. We will refer to this parameter as *RetryLimit*, which denotes the maximum number of times that a data-link layer frame is retransmitted.

We eliminate wireless losses by assigning a very large value to the *RetryLimit* parameter at the mobile nodes. In the uplink direction, the *RetryLimit* parameter affects only TCP segment retransmissions. Moreover, it does not affect other flows from the same node, since if a single flow fails to transmit then no other flow from the same node will be able to transmit. Hence, using a large *RetryLimit* parameter, it is highly improbable for a data-link layer frame to be dropped, even if packet errors are bursty, and packet losses are, almost, only due to congestion, as in wireline networks. In

other words, we hide from the TCP sender the unreliability of the wireless channel. We will refer to this modification of the 802.11 protocol as the infinite-RetryLimit (inf-RL) enhancement. However, persistent local retransmissions introduce an unpleasant effect; packets being locally retransmitted can experience significantly larger delay than the packets that did not undergo retransmission. In bursty loss environments these delay variabilities are expected to be fairly large and can lead to spurious timeouts.

4.2.2 Adaptive algorithm to absorb high delay variability

Our goal is to develop an algorithm that absorbs high delay variability caused by local retransmissions at the data-link layer. The algorithm's objective is to dynamically inject artificial delay to ACK packets, so that we indirectly influence the RTO calculation and prevent it from expiring. An important observation is that the size of the delay spike reflects the packet's delay due to local retransmissions. Thus, this information, considering the burstiness of errors in the wireless channel, provides the intuition of how much delay we should inject in order to avoid potential spurious timeouts. The packets which are selected to be delayed are TCP acknowledgement (ACK) packets that reach the access point from the wired nodes. The access point implements the functions needed for our adaptive delay injection method.

For the purposes of the algorithm, we define packet delay (*pdelay*) as the interval from the time the mobile TCP sender transmits a packet until the time the corresponding TCP acknowledgement reaches the access point. This interval is calculated from the access point's local time minus the TCP acknowledgement's timestamp[1], and includes propagation delays, queuing delays, and delays spent on local retransmissions. Fig. 4.1 depicts the *pdelay* variable, which is calculated for every incoming TCP acknowledgement at the access point.

Also, let average delay (*avdelay*) be the exponentially smoothed average of *pdelay*. When there are no errors at the wireless channel, i.e. there are no link-layer retransmis-

[1]The TCP timestamp needs to be enabled. Also, this assumes that clocks are synchronized. If this is not the case, the access point can estimate the time lag, and consider it in the calculation.

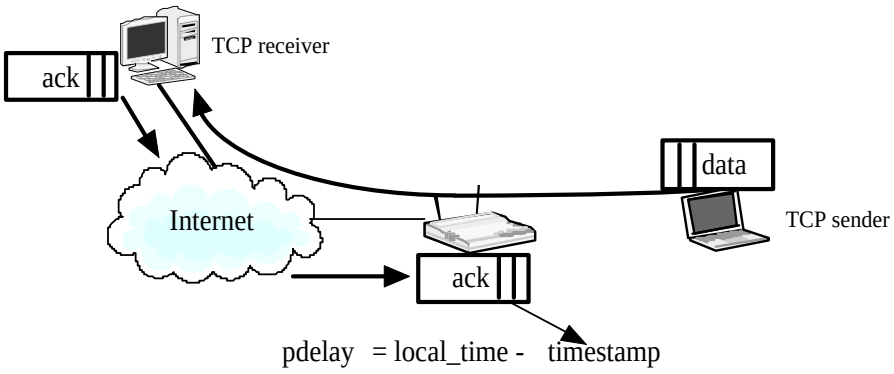


Figure 4.1: Calculation of the *pdelay* variable

sions, *avdelay* will include only the propagation and queuing delays. It is,

$$avdelay[k] = (1 - a) * avdelay[k - 1] + a * pdelay[k] \tag{4.1}$$

where k is the $k - th$ incoming TCP ACK at the access point and a is given the value of $\frac{1}{8}$. So, for a packet that undergoes local retransmission, *pdelay* will include the time spent on these retransmissions. Hence, the difference of *pdelay* and *avdelay* gives a good estimate of the time spent on local retransmissions, and therefore suggests the delay we need to inject in order to avoid spurious timeouts.

In order to identify delay spikes that can lead to spurious timeouts, we apply a formula similar to the TCP RTO calculation formula, as suggested in [8], on the variable *pdelay* at the access point. So, for the $k - th$ incoming TCP acknowledgement at the access point, we have:

$$S[k] = (1 - a) * S[k - 1] + a * pdelay[k] \tag{4.2}$$

$$V[k] = (1 - b) * V[k - 1] + b * (|pdelay[k] - S[k]|) \tag{4.3}$$

$$dthresh = S[k] + c * V[k] \tag{4.4}$$

This formula calculates the *dthresh* variable as the sum of the smoothing average of *pdelay* (S) plus its variance (V). Typical values for a, b and c are $\frac{1}{8}, \frac{1}{4}$ and 4

respectively, as suggested in [8]. Identification of the delay spike is crucial for our algorithm since it indicates when delay should be injected. The *dthresh* is compared to *pdelay* for every incoming acknowledgement at the access point in order to detect a delay spike. Once a delay spike is detected, the delay D to inject is proportional (defined by $delay\_factor$) to the size of the delay spike detected, i.e. *pdelay* minus *avdelay*. Hence

$$D = \frac{pdelay - avdelay}{delay\_factor}$$

The algorithm, also, includes a linear reduction parameter ($subthres$[2]) of the injected delay to avoid needlessly delaying packets when delay spikes are infrequent. A high level description of the proposed algorithm, which runs for every ACK packet received by the access point and travelling towards the mobile nodes, is shown below.

Algorithm 1 Adaptive delay injection algorithm

1: **for all** incoming ACKs **do**
2: $pdelay = (ACK \rightarrow timestamp) - local\_time$
3: **if** $pdelay > dthresh$ **then**
4: /*delay spike detection*/
5: $D = \frac{(pdelay - avdelay)}{delay\_factor}$
6: **else**
7: $inject\_delay(D)$
8: **end if**
9: $D = D - subthresh$
10: $avdelay = (1 - a) * avdelay + a * pdelay$
11: $update(dthresh)$
12: **end for**

The $inject\_delay()$ function adds delay D to the first ACK packet after the delay spike is detected. All subsequent ACK packets that arrive within time D of the delayed ACK are transmitted after it (i.e., no additional delay is added). ACKs that arrive after time D of the first delayed ACK are again delayed by time D. The value of D

[2]Detailed explanation of the $delay\_factor$ and $subthresh$ parameter selection is included Chapter 5.

decreases linearly with each ACK that does not trigger a delay spike detection. Finally, the *update*() function calculates the *dthresh* value according to equations (2.1), (2.2) and (2.3).

The delay injection procedure is depicted in Fig. 4.2. Suppose that at some time, the acknowledgment packet a arrives at the access point. Algorithm 1 detects a high delay variability for ACK a and calculates the value D of the delay to be injected; ACK a is transmitted without any delay. Afterwards, ACK b arrives at the access point; this ACK is delayed by D. Subsequent ACKs (such as ACK c) that arrive before time D elapses are not further delayed, hence, are transmitted immediately after packet b. An ACK packet (d) that arrives after the interval D is delayed by an interval $D - subthersh$, as the algorithm suggests. This procedure continues until either delay D is decreased to 0 or a new delay spike is detected and the value D is re-calculated.

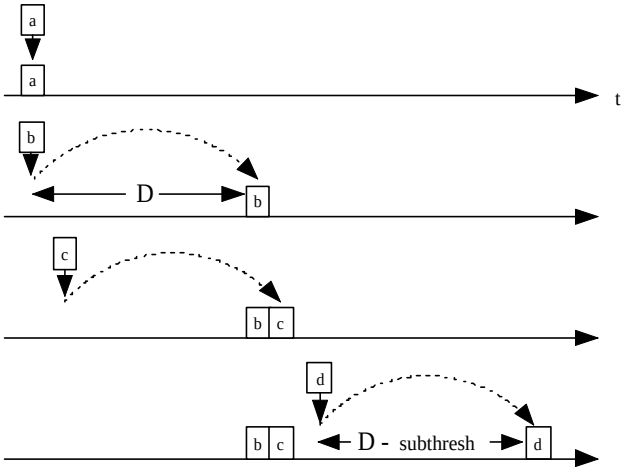


Figure 4.2: Delay Injection Procedure

The reason why we implement the delay injection algorithm for ACK packets reaching the access point from the wired part of the LAN, and not for TCP segment that reach the access point by the mobile nodes, is that ACKs are just one hop away from their final destination, i.e. the TCP sender that lies at the mobile nodes. Recalling that *pdelay* of a TCP ACK equals to the RTT of the corresponding TCP segment minus the propagation delay of the wireless path, the delay spike detection mechanism, almost,

simulates the TCP sender's timeout detection mechanism. Therefore, we can be sure enough when a delay variability will also be experienced as a timeout by the TCP sender. This renders our adaptive delay injection algorithm adaptable to networks with different propagation delays, since the case where a delay variability leads to a spurious timeout depends on the delay of the path.

4.3 Considering downlink traffic

In the previous section, we presented the methodology for suppressing wireless losses and absorbing high delay variabilities, when data packets were travelling from the mobile nodes to the access point and ACKs to the opposite direction.

Suppressing wireless data packet losses at the uplink suggests assigning a large value at the *RetryLimit* parameter of 802.11 at the mobile nodes. Though, suppressing packet losses at the downlink presents different characteristics. Applying a strong local error recovery mechanism at the access point (by assigning a large value at the *RetryLimit* parameter) may have a negative impact in a scenario where multiple wireless nodes, with different radio channel quality, are being served by the same access point. Suppose, that there is a wireless LAN with more than one participating wireless nodes and that one of them has very low channel quality, i.e. a obstacle in front of the mobile node is intercepted. Because of the large *RetryLimit*, the access point will attempt several retransmissions of the same packet (destined to the mobile node with the low channel quality) without success. Hence, other nodes, that may have a good channel, will be prevented from transmitting.

In order to avoid situations, where the robust local error recovery mechanism blocks packets destined to all mobile nodes, we propose a modified round-robin serving mechanism at the access point, shown in Fig. 4.3. The packet serving mechanism, initially, suggests that there is a single queue for every destination mobile node; Queue 1, Queue 2, ..., Queue N for destination mobile nodes 1, 2, ... N, respectively. Packets destined to each one of the destination nodes are served in a round robin fashion. More-

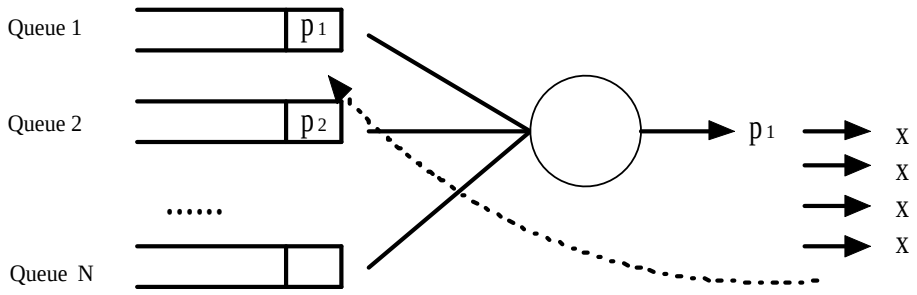


Figure 4.3: The modified round robin serving mechanism

over, every time the access point fails to transmit a packet to a destination mobile node, then the unsuccessfully transmitted packet is not dropped, but is put back at the front of the corresponding queue. For example, as shown in Fig. 4.3, suppose that packet p_1 destined to node 1 is set for transmission by the access point. Due to low channel quality for node 1 the packet transmission fails for four consecutive times. Then, instead of the packet being dropped it is put back to the front of Queue 1 and the access point carries on with the next queue (Queue 2) and, hence, transmits packet p_2. This way, the access point implements a strong local error recovery mechanism, without blocking packets that may have a good channel quality. Then, as with uplink data traffic, the high delay variabilities, introduced by the local retransmissions, can be alleviated with the aid of our adaptive delay injection algorithm.

Chapter 5

Experimental Evaluation

In this chapter, we present the experimental results from the application of our method in single and multihop wireless networks with Internet infrastructure, based on 802.11. All experiments were conducted using the Network Simulator 2 (NS-2) [11]. We compared three different approaches:

- Standard 802.11 (st-802.11): We used the standard 802.11 protocol implementation without any modification throughout the experiments. The default values for the *ShortRetryLimit* and *LongRetryLimit* parameters are 7 and 4, respectively.
- Infinite-*RetryLimit* enhancement (inf-*RL*): This modification to the standard 802.11 considers assigning a very large value for both the *ShortRetryLimit* and *LongRetryLimit* at the mobile nodes. Practically, we assigned the value 50 to both of the parameters. In the remainder of this report both will be referred to as *RetryLimit* parameter.
- Delay injection method (DI-method): This method combines the inf-*RL* enhancement and the adaptive delay injection algorithm, presented in Chapter 4.

We tested all three methods in wireless topologies with different network characteristics, such as propagation delays, packet error rates (PERs) and number of participating wireless nodes. Furthermore, in multihop network scenarios, we conducted experiments

with different number of hops. Note, also, that in the experiments we only considered uplink data traffic.

5.1 Performance metrics

The network performance metrics considered in our experiments were the end-to-end throughput and the useful transmission ratio[1]. The end-to-end throughput of a TCP flow is equal to the ratio of the total amount of transmitted data volume at application layer to the duration of the corresponding transmission. Hence,

$$\text{end-to-end throughput} = \frac{\text{total data transmitted at application layer}}{\text{duration of data transmission}} \tag{5.1}$$

The useful transmission ratio is defined as the ratio of the transmitted useful data to the total amount of data transmitted over the network. The latter amount of data includes unnecessary segment retransmissions by the TCP sender, due to, falsely, triggered congestion avoidance mechanisms. So, it is:

$$\text{useful transmission ratio} = \frac{\text{useful bytes transmitted}}{\text{total bytes transmitted}} \tag{5.2}$$

We measured the useful transmission ratio in order to quantitatively determine the overhead imposed by the unnecessary retransmissions because of the spurious timeouts.

5.2 Infrastructure networks experiments

The network topology used throughout the infrastructure network experiments is depicted in Fig. 5.1. The network consists of a number of mobile nodes, an access point and a number wired nodes. The mobile nodes act as TCP senders and are connected to the wired TCP receivers through the access point. Also, the wired TCP receivers are connected to the access point through an intermediate node. The TCP implementation

[1] In multiuser environments we consider the aggregate of this metric.

used is TCP Sack [12]. The wireless protocol used is 802.11b operating at 11 Mb/s and distance of the mobile nodes from the access point was set to 200 m. Finally, the wired links have a capacity of 100 Mb/s, i.e. there is no bottleneck in the wired part of the network.

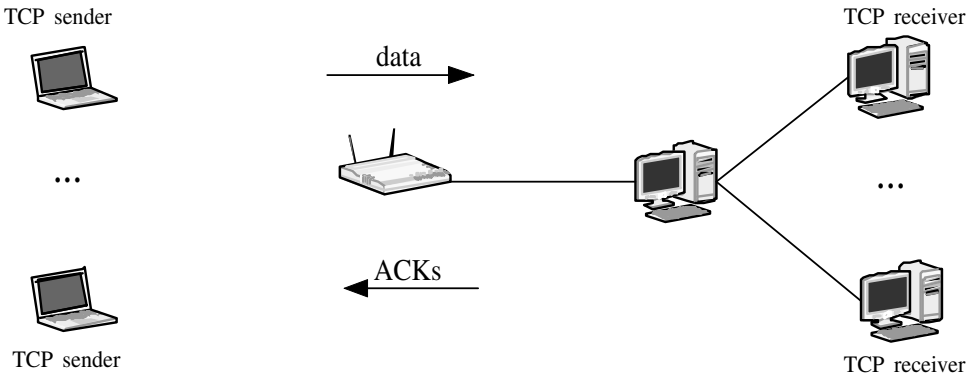


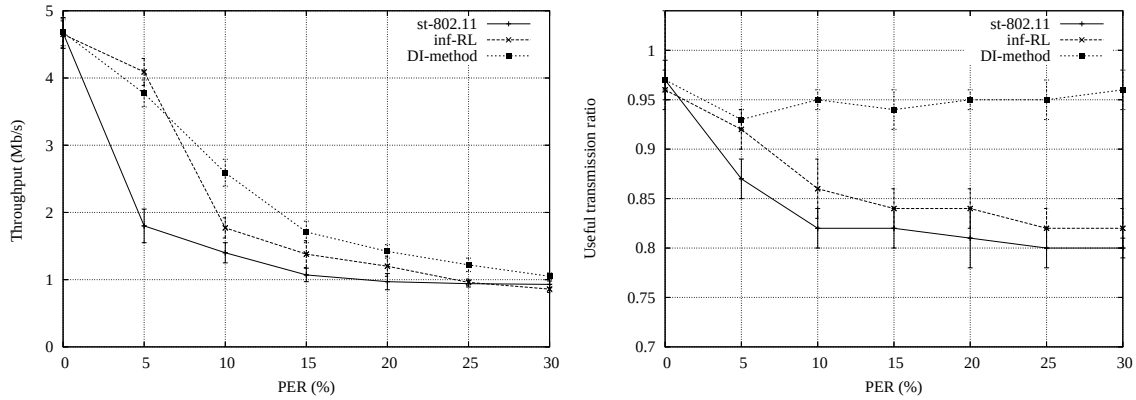
Figure 5.1: Infrastructure Network Topology

In the experiments conducted, we consider long-lived FTP traffic. The duration of a single experiment is 140 seconds. Each FTP flow starts randomly within the interval [0,10] seconds. The measurements are taken after the 20th second, in order to exclude from our results the time spent on the connection establishment and slow start phase. To model bursty losses at the wireless channel, we use a simple error model that introduces consecutive packet losses, e.g. in a 10% loss scenario 10 out of 100 packets are lost, consecutively.

The inf-RL enhancement is implemented by assigning a large value (equal to 50) for the $ShortRetryLimit$ and $LongRetryLimit$ parameter of the 802.11 protocol. Moreover, our algorithm was implemented at the data-link layer of the access point, and requires access to the TCP/IP headers. Finally the timestamp option was enabled for every TCP session.

5.2.1 Single user experiment

In this subsection, we compare the performance of the three approaches in a single flow scenario. In Fig. 5.2, we show the TCP throughput and the useful transmission ratio as a function of the packet error rate (PER) for one TCP flow, when the wired propagation delay (w_prop) is 15 ms.



(a) Throughput (b) Useful transmission ratio

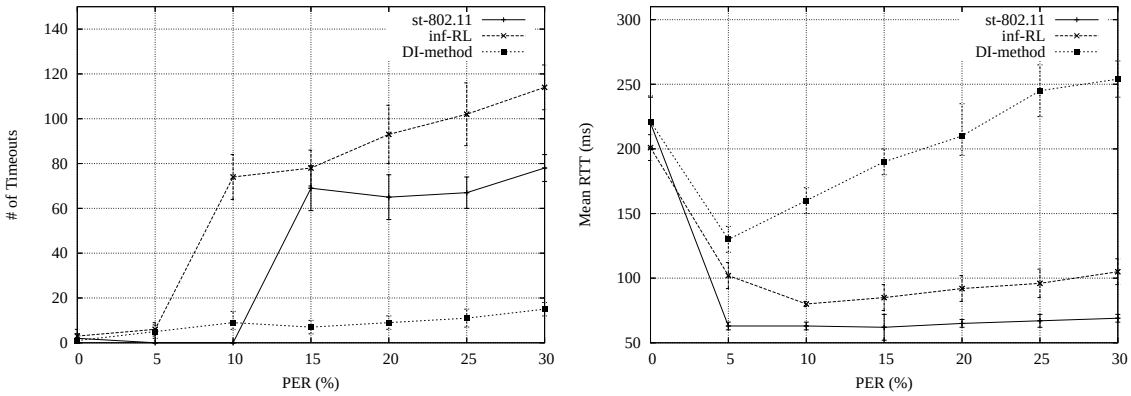
Figure 5.2: End-to-end TCP throughput and useful transmission ratio as a function of PER: 1 TCP flow, 15 ms wired propagation delay

As shown in Fig. 5.2(a), TCP over standard 802.11 experiences poor throughput due to packet losses being misinterpreted as congestion signals. The inf-RL enhancement solves the above problem, since packets are no longer dropped at the wireless channel, but introduces high delay to the packets that are retransmitted at the data-link layer, especially when PER is fairly large. High delay variability results in spurious timeouts and, therefore, severe throughput degradation. The above two problems are addressed by our method, which performs significantly better than the other two methods when PER is above 10%. For example, our algorithm presents throughput improvements of 85% and 46% in comparison to the st-802.11 and inf-RL enhancement respectively, when PER is 10%. When PER is 5%, the improvement of our method is smaller than

the inf-RL enhancement, though still better than st-802.11 (improvement of 109%). This is because a low PER (5%) does not impose much delay variability to the locally retransmitted packets so as to generate spurious timeouts. In this case, our algorithm sometimes decides to inject delay to ACK packets, that eventually does not have a positive impact on TCP performance, but rather a slightly negative one, resulting in a throughput reduction of 8% in comparison to the inf-RL enhancement.

In Fig. 5.2(b) we show the useful transmission ratio as a function of the PER. The 802.11 standard suffers from poor useful transmission ratio because, firstly, the TCP sender experiences *ordinary* timeouts; we will refer to as *ordinary* timeouts the timeouts caused by multiple packet losses at the data-link layer. In addition, packet losses at the data-link layer trigger duplicate ACKs that lead to segment retransmissions through the congestion avoidance mechanism invocation. Hence, a single TCP segment is transmitted more than once by the TCP sender. The reduction in the useful transmission ratio increases as the PER increases, due to the increasing number of packet losses and timeouts. The inf-RL enhancement, also, presents low useful transmission ratio due to spurious timeouts, triggered by high delay variabilities. Our method, improves the useful transmission ration, since it prevents ordinary timeouts, duplicate ACKs and spurious timeouts. Ordinary timeouts and duplicate ACKs are prevented with the aid of the strong local error recovery mechanism, while spurious timeouts with the aid of the adaptive delay injection algorithm. For example, when PER is 30% the increment in the useful transmission ratio of our method in comparison to the st-802.11 and the inf-RL enhancement is 18% and 17%, respectively.

In Fig. 5.3 we present the number of timeouts and the impact on the RTT of each approach studied as a function of the PER, in the same single flow experiment. In Fig. 5.3(a), we see that, as explained above, the our method achieves significant reduction of the number of timeouts, compared to both the standard 802.11 and the inf-RL enhancement. Moreover, in Fig. 5.3(b), we see the impact of the delay injection method on the round trip time of the TCP session. Our method increases the round trip time of the session due to the artificial delay that is injected to the TCP ACKs. Note



(a) # of Timeouts (b) mean RTT

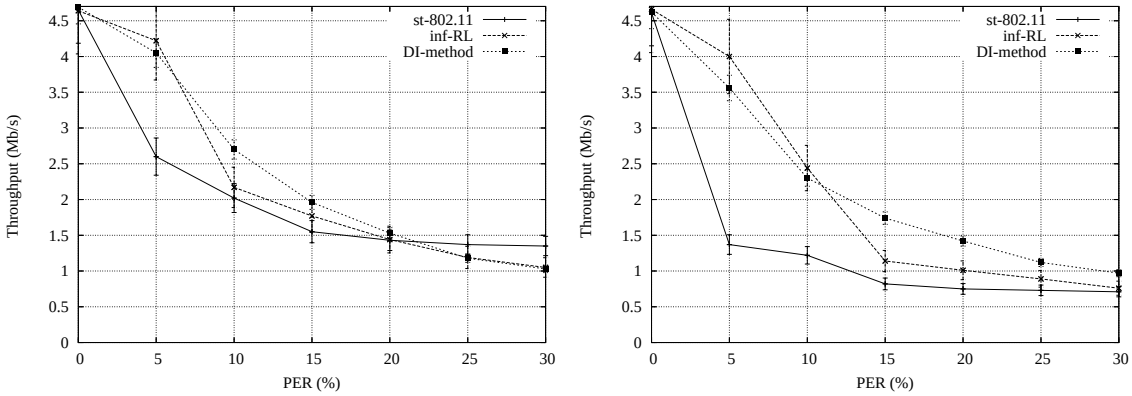
Figure 5.3: # of Timeouts and mean RTT as a function of PER: 1 TCP flow, 15 ms wired propagation delay

that the increase in RTT is bigger when PER is larger, because the delay that has to be injected in order to avoid spurious timeouts is larger. In the next two subsections, we see the impact of the delay of the path and the number of mobile nodes on TCP performance and show that avoiding spurious timeouts is not always throughput-beneficial.

5.2.2 Experiments with different propagation delays

In this section, we study the impact of different propagation delays on each of the three approaches. The propagation delay of a TCP session's path is straightforwardly related to the end-to-end throughput, which is inversely proportional to the RTT of the path. Note, also, that a larger propagation delay suggests a larger bandwidth delay product (BDP) and, therefore, larger values for the *cwnd* of the TCP session. When the *cwnd* is large, the impact of a spurious timeout on TCP performance is greater since the *cwnd restoration phase* will occur more slowly. Moreover, the larger the propagation delay is, the longer the *cwnd restoration phase* lasts, since the *cwnd* is incremented more slowly.

In Fig. 5.4, we show the throughput as a function of PER, for two wired differ-



(a) Throughput, $w\_prop = 10$ ms

(b) Throughput, $w\_prop = 20$ ms

Figure 5.4: End-to-end TCP throughput: 1 TCP flow, 10 ms and 20 ms wired propagation delay, respectively

ent propagation delays at the wireline part; w_prop is 10 and 20 ms, respectively. In Fig. 5.4(a), we can see that our method performs better than both the other two only when PER is 10%, 15% and 20%. When PER is lower than 10% then, as in the former experiment, no spurious timeouts take place at the TCP sender. On the other hand, when PER is greater than 20%, the 802.11 standard outperforms the other two approaches. This is attributed to the two following reasons: (i) because of the high PER ($> 20\%$) the local error recovery mechanism adds significant delay to packets that undergo retransmission, which stay in buffers for a respectful amount of time, and (ii) the small delay of the path (only 10 ms) renders end-to-end retransmission of lost packets more beneficial than have them locally retransmitted. Thus, when the end-to-end delay is low and the PER is high, it is better to leave packet retransmission to the end-to-end mechanism.

Fig. 5.4(b), illustrates the results when the wired propagation delay is 20 ms. We see that our method improves performance, in comparison to the inf-RL, when PER is greater than 15%. When PER is 10%, in contrast to the experiment where the wired propagation delay was 15 ms, our method does not outperform the inf-RL enhancement. This is attributed to the fact that the large propagation delay suggests a

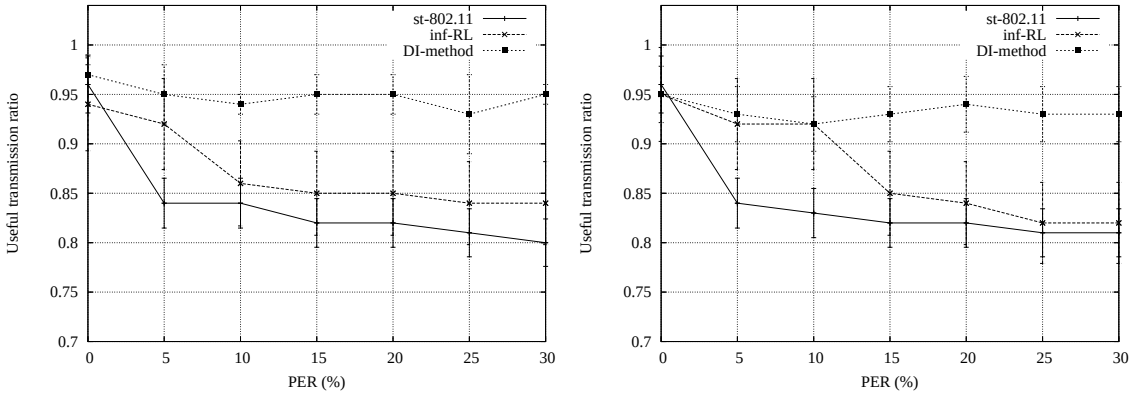
larger BDP for the path and, hence, more packets are injected to the network because the *cwnd* obtains greater values. Therefore, due to the great number of outstanding packets, larger inter-packet variances are introduced, rendering the TCP sender's RTO more robust against delay variabilities. Thus, less spurious timeouts take place. This observation is also shown in Table 5.1, where we show the number of timeouts for different wired propagation delays, when PER is 20%. Moreover, from Fig. 5.2(a) and Fig. 5.4(b), we see that the improvements achieved with our method are higher when the wired propagation delay is larger (20 ms), when there are spurious timeouts. Hence, when the propagation delay is larger, the negative impact of spurious timeouts is more crucial and, therefore, avoiding them is more beneficial.

Table 5.1: Number of timeouts as a function of propagation delay, when $PER = 20\%$

	10ms	15ms	20ms	30ms	50ms	80ms
st-802.11	96	65	50	39	24	16
$inf - RL$	158	93	77	49	15	5
$DI - method$	3	9	3	7	4	4

In Fig. 5.5, we see the impact of the propagation delay on the useful transmission ratio. It is worth mentioning that in the case where st-802.11 outperforms the other two methods ($w\_prop = 10\ ms, PER \geq 25\%$), our method achieves improvements in the useful transmission ratio, because it avoids retransmission triggered by spurious timeouts, ordinary timeouts and reception of three duplicate ACKs. Moreover when w_prop is 20 ms and PER is 10% there is no useful transmission ratio improvement of our method against the inf-RL enhancement since no spurious timeouts occur at the TCP sender.

Finally, the overall impact of the wired propagation delay on TCP throughput for different PERs is shown in Fig. 5.6.



(a) Useful transmission ratio, $w\_prop = 10$ ms (b) Useful transmission ratio, $w\_prop = 20$ ms

Figure 5.5: Useful transmission ratio: 1 TCP flow, 10 ms and 20 ms wired propagation delay, respectively

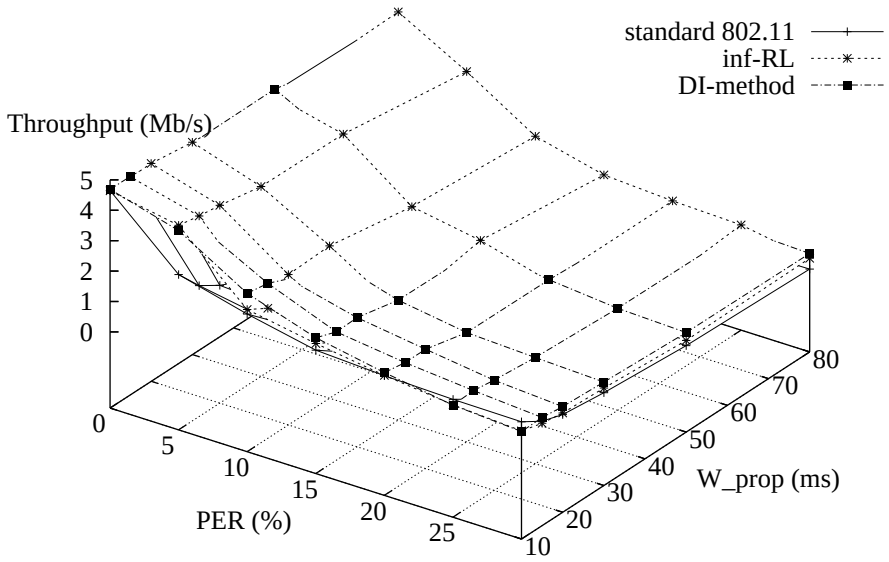
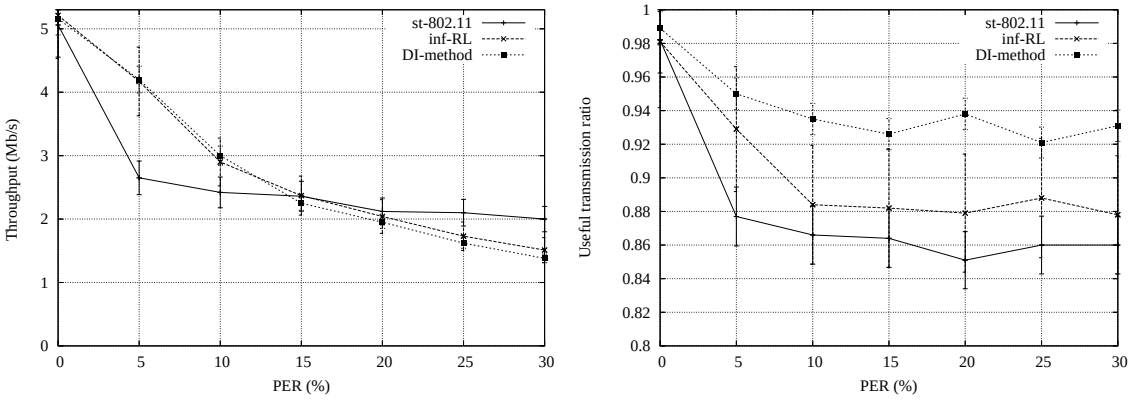


Figure 5.6: Impact of the wired propagation delay (w_prop) on TCP throughput for different PERs. The 95% confidence interval is within +/- 9% of the values shown

5.2.3 Experiments with multiple users

In this section we present the results of the experiments conducted with more than one participating mobile nodes. Each participating mobile initiates one TCP session

with a destination node at the wired part of the network.

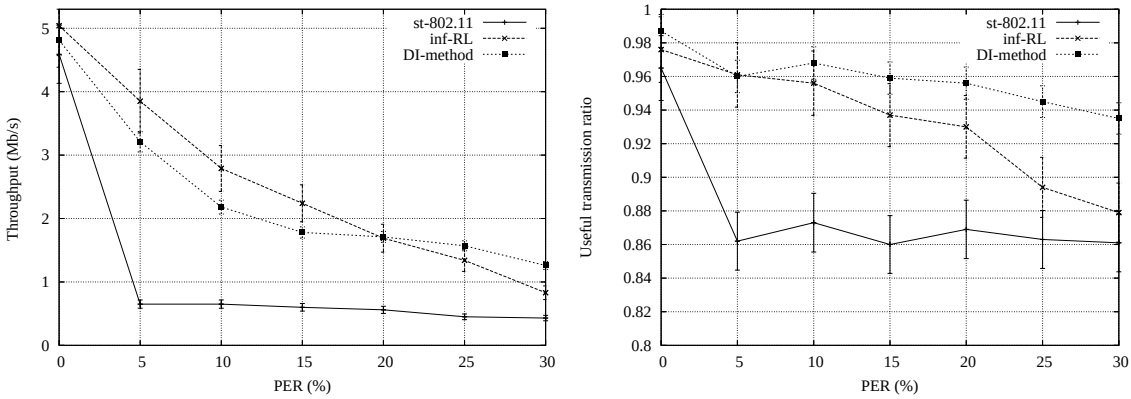


(a) Throughput (b) Useful transmission ratio

Figure 5.7: End-to-end TCP throughput and useful transmission ratio: two TCP flows, 15 ms wired propagation delay

In Fig. 5.7, we show the aggregate throughput and useful transmission ratio, respectively, as a function of the PER for two participating mobile nodes. The wired propagation delay is $15ms$. As shown in Fig. 5.7, when PER is below 15%, both our method and the inf-RL enhancement outperform the st-802.11. Our method does not outperform the inf-RL enhancement, when PER is 10%, in contrast to the one flow experiment, because the participation of two mobile nodes limit the optimal *cwnd* of each flow; hence, the *cwnd restoration phase* is shorter and avoiding spurious timeouts does not improve performance. Moreover, when PER is above 20%, the 802.11 standard outperforms the other two approaches. This is because, due to the the high $PERs$, the two TCP flows experience significant data-link layer contention, since each node continuously attempts to retransmit lost packets and, hence, prevents the other node from transmitting. This observation, in conjunction with the low wired propagation delay, renders end-to-end retransmission more beneficial. However, as shown in Fig. 5.7(b), even when $PER > 20\%$, our method improves the useful transmission ration, because it prevents ordinary timeouts, unnecessary retransmissions due to duplicate ACKs and spurious timeouts.

The same experiment was conducted with a larger wired propagation delay, i.e. 80 ms; the results for the aggregate throughput and useful transmission ratio are shown in Fig. 5.8. For all PER values, our method performs significantly better than the 802.11 standard. When $PER \leq 20\%$ the inf-RL enhancement outperforms our method. This is, again, attributed to the fact that low $PERs$ do not impose a lot of spurious timeouts in a large delay path. However, when $PER \geq 25\%$, plenty of spurious timeouts are introduced and, hence, our approach yields significant throughput improvements. Finally, as shown in Fig. 5.8(b), our method achieves significant useful transmission ratio improvements, when $PER > 20\%$, i.e. plenty of spurious timeouts are introduced.



(a) Throughput (b) Useful transmission ratio

Figure 5.8: End-to-end TCP throughput and useful transmission ratio: two TCP flows, 80 ms wired propagation delay

Finally, in Fig. 5.9, we show the overall impact of the number of mobile nodes on the aggregate throughput for different wired propagation delays. The PER is 20%.

5.3 Multihop networks experiments

In this section, we present the results of our method's application to multihop wireless networks connected to wired Internet infrastructure. As in the infrastructure network experiments, the TCP sender lies in a mobile host while the TCP receiver lies in the Internet. The wireless protocol used is 802.11 at 11 Mb/s, while the TCP

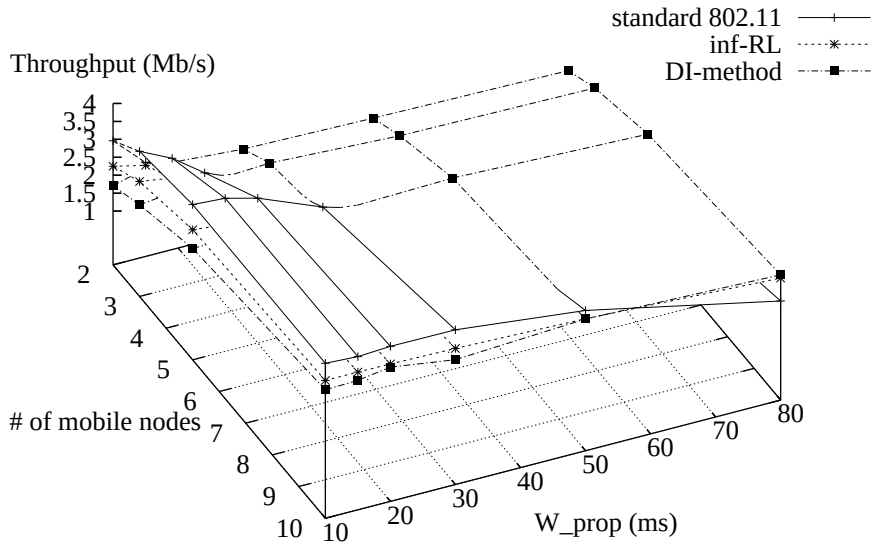


Figure 5.9: Aggregate throughput as a function of the number of mobile nodes for different PERs, when $PER = 20\%$. The 95% confidence interval is within +/- 11% of the values shown

implementation is, again, TCP Sack. Finally the wired link's capacity is 100 Mb/s. The mobile nodes are within a distance of 200 m. The network topology used throughout the experiments in depicted in Fig. 5.10. We conducted experiments with different propagation delays, packet error rates, and number of hops at the multihop part of the network.

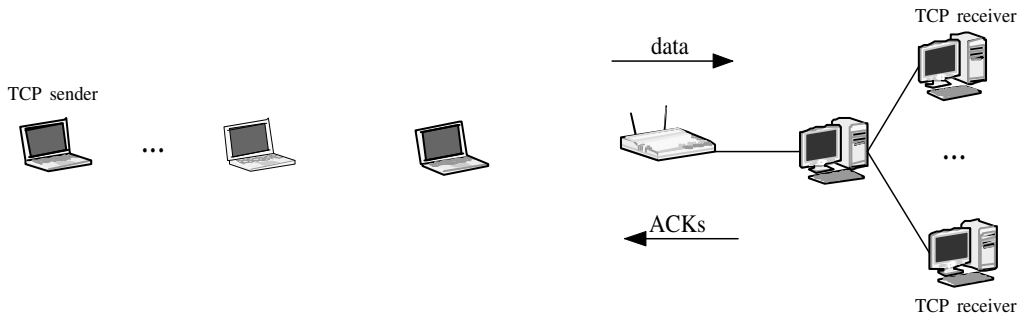
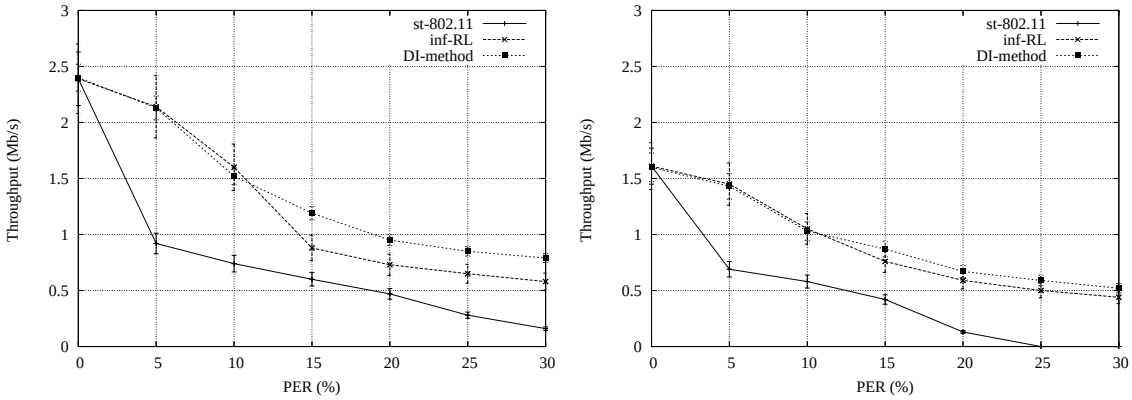


Figure 5.10: Multihop network topology



(a) Throughput: Two hops

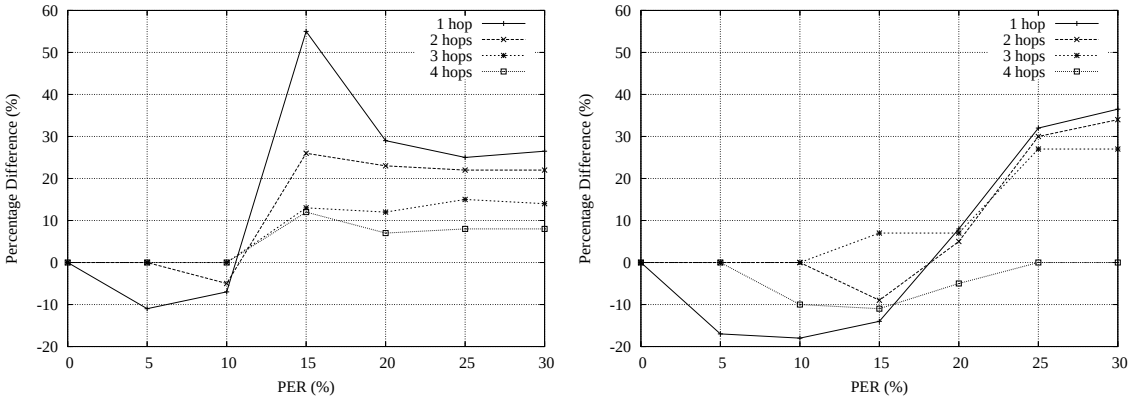
(b) Throughput: Three hops

Figure 5.11: End-to-end TCP throughput as a function of PER for two and three hops, when the wired propagation delay is 20 ms

Fig. 5.11 shows the TCP throughput as a function of the PER for a two and three hop multihop network, respectively, when the wired propagation delay is 20 ms. The results are qualitatively the same with infrastructure wireless networks. Both the inf-RL and our method perform better than the standard 802.11 protocol for all PER values, while our method outperforms the inf-RL enhancement in the presence of spurious timeouts; i.e. when $PER \geq 15\%$. Moreover, from Fig. 5.11(a) and (b) we see that the improvement of our method in comparison to the inf-RL is higher in the case of two hops compared to the case of three hops. This is because, due to the increasing number of hops, the optimum *cwnd* of the TCP session is smaller and, hence, the *cwnd restoration phase* is shorter. In this case, avoiding spurious timeouts becomes less beneficial.

In order to compare our method and the inf-RL enhancement, we show the percentage difference in throughput of our method and the inf-RL as function of PER for different numbers of hops. This comparison is show in Fig. 5.12, for two different values of the propagation delay at the wireline part, i.e. 20 and 50 ms.

As shown from Fig. 5.12(a) and (b), the improvements of our method in comparison to the inf-RL enhancement become smaller as the number of hops increases. This is attributed to the fact that as the number of hops increases the optimal *cnwd*



(a) Throughput, $w\_prop = 15\ ms$ (b) Throughput, $w\_prop = 50\ ms$

Figure 5.12: Percentage difference of our method and the inf-RL as a function of the number of hops, for wired propagation delays 20 ms and 50 ms

window of the TCP flow becomes smaller. For example, when the number of hops is four, the improvements of our method are almost negligible. As far as the performance improvement "knee" that is observed (in (a) it is observed when $PER \geq 15\%$, while in (b) when $PER \geq 20\%$), it depends on the propagation delay at the wireline part. As in the infrastructure experiments, the greater the propagation delay at the wired part is the greater the PER has to be in order for spurious timeouts to be produced.

5.4 Algorithm parameter selection

Our delay injection algorithm has two parameters that need to be defined: the $delay\_factor$ and the $subthresh$. In this section, we show the appropriate selection of these two parameters, in an infrastructure network scenario.

5.4.1 Delay factor ($delay\_factor$) selection

As introduced in Algorithm 1, the $delay\_factor$ parameter reflects how much delay is to be injected in proportion to the size of the delay spike. Fig. 5.13 shows the effect of $delay\_factor$ parameter on TCP performance and, hence, suggests what is the

appropriate value to be used. In this experiment, only one mobile user participated in the network and, also, the wired propagation delay was 15 ms.

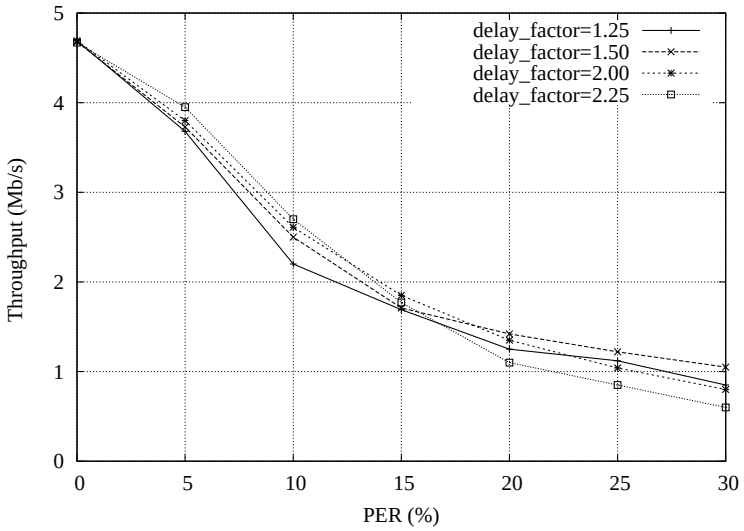


Figure 5.13: Delay_factor parameter selection. The 95% confidence interval is within +/- 8% of the values shown

A smaller $delay\_factor$ results in a larger delay being injected, hence decreases the likelihood of delay spikes. Though, when then the $delay\_factor$ is really small (e.g. 1.25) too much delay is injected leading to significant increment of the RTT of the TCP session and, hence, throughput reduction. On the other hand, a larger $delay\_factor$ value results in more significant throughput improvement in low PER scenarios, e.g., when $delay\_factor = 2.25$ and $PER = 10\%$, but this is not the case when PER is higher. We choose a rather small value (1.5) to guarantee that less spurious timeouts will occur.

5.4.2 Subtraction threshold ($subthresh$) selection

In contrast to $delay\_factor$, $subthresh$ does not directly affect TCP's performance. Its goal is to mitigate the negative impact of delay injection in scarce loss environments. Suppose that in the wireless environment a burst of corrupted packet occurs; our algorithm will detect the delay spike and would inject delay according to Algorithm 1.

Without the presence of the *subthresh* parameter the access point would continue delaying ACKs despite the fact that there is no other delay variability. In this case the injected delay would be unnecessary and would result in throughput degradation. The *subtraction threshold* parameter decreases the delay to be injected to every incoming TCP acknowledgement by the value of *subthresh*. Therefore after several incoming ACKs, and if no other delay variability takes place, the delay to be injected would be zero. Hence, ACK packets will no longer be, unnecessarily, delayed.

The *subthresh* is selected as follows: the access point counts the number of packets that are interjected between two delay spikes (*pkt_count*); supposing that delay D is chosen to be injected, *subthresh* is assigned the value $\frac{D}{5*pkt\_count}$. With such a value, if in an interval containing $5 * pkt\_count$[2] packets no delay spikes occur, then the value of the injected delay is reduced to zero.

5.5 Comparison to other methods

In this section we present the results of the comparison of our method to other methods presented in the bibliography.

5.5.1 Comparison to static delay injection

We compared our adaptive delay injection method (DI-method) to the static delay method (static DI-method). In the latter method, the magnitude of the delay to be injected is chosen statically, while the packets to be delayed are selected according to the delay spike detection mechanism described in Algorithm 1. We conducted experiments with different PERs in order to show the relation each of the two methods to this characteristic.

In Fig. 5.14, we see the end-to-end throughput as a function of the PER, when the wired propagation delay is 15 ms. The magnitude of the delay to be injected in the static-DI method is 60 ms, i.e. every time the delay spike detection mechanism

[2]The value 5 is optional, denoting the maximum distance, in packets, of two consecutive delay spikes.

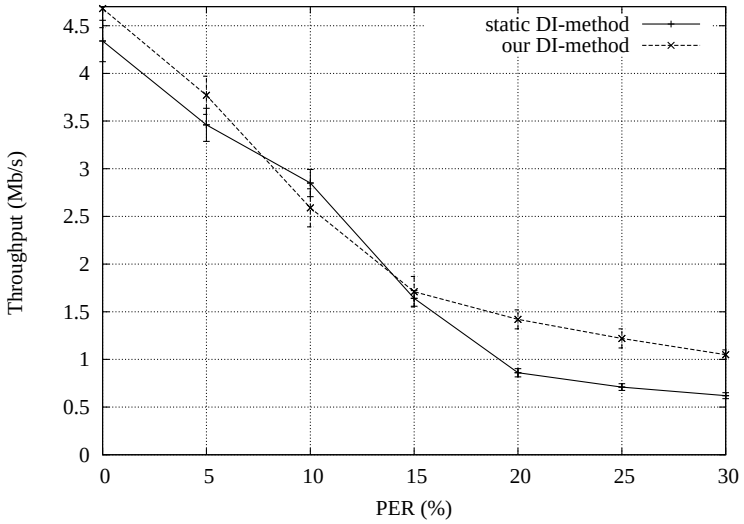


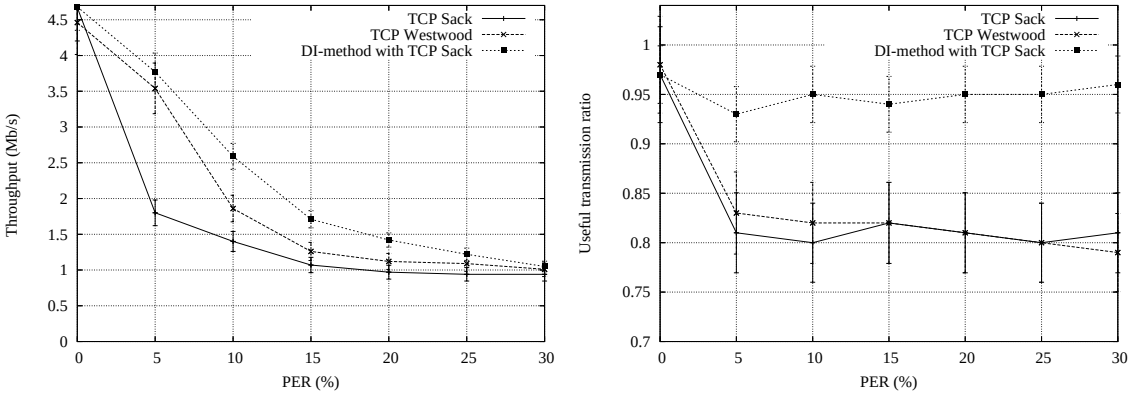
Figure 5.14: Comparison of our method with static delay injection for different PERs, when the wired propagation delay is 15 ms

of Algorithm 1 detects a delay spike, then ACKs are delayed by 60 ms, as illustrated in Fig. 4.2. As we can see in Fig. 5.14, the static delay injection method outperforms ours only when PER is 10%, hence, the selection for delay to be injected according to our algorithm is less appropriate. On the other hand, we see that for all other PER values, our method outperforms the static delay injection method, denoting that the static selection for the delay in the static-DI method is does not adapt to different packet error rates.

5.5.2 Comparison to TCP Westwood

Next, we compare our method (with TCP Sack implementation at the TCP sender) with Westwood TCP [13] and TCP Sack without the proposed method. The results for the TCP throughput and the useful transmission ratio as a function of PER are shown in Fig. 5.15(a) and (b).

As we can see in Fig. 5.15(a), TCP Westwood achieves better throughput than TCP Sack because of the efficient *cwnd* and *ssthresh* restoration procedure after an unnecessary *cwnd* reduction. This way the *cwnd* restoration phase is dramatically re-



(a) Throughput (b) Useful transmission ratio

Figure 5.15: Comparison of DI-method to TCP Westwood

duced and, therefore, the throughput improves. On the other hand, TCP Westwood's performance is lower than the proposed method's, because it still experiences timeouts and *cwnd* reductions due to spurious timeouts. The unnecessary retransmissions that also take place with TCP Westwood are also depicted in Fig. 5.15(b). TCP Westwood achieves the same goodput as TCP Sack, which is significantly less than our method's.

Chapter 6

Related Work

6.1 Improving TCP performance over wireless links

Several methods that improve TCP throughput have been proposed in the bibliography over the recent years. In general, the approaches that study the performance of TCP in wireless networks lie in one of the following three categories, as presented in [10]: end-to-end mechanisms, link-layer approaches and split connections.

6.1.1 End-to-end mechanisms

This category includes studies that propose modifications that take place at the end hosts, either by implementing a new protocol or by modifying the mechanisms at the sender or the receiver. Modifications of the TCP protocol include TCP protocol extensions, such as TCP Newreno [14] and TCP Sack [12]. These two protocols use an enhanced cumulative acknowledgement scheme in order to recover from multiple packet losses within a single transmission window. A TCP sender using the ECN mechanism [15] distinguishes between losses due to congestion and losses due to packet corruption and therefore does not falsely respond to congestion or packet losses. Although ECN was originally introduced for wired networks, F. Peng *et al.* [16] propose a method to effectively use this mechanism in wireless environments as well. As far as

end-to-end methods are concerned, besides needing changes to the end hosts, they can also require longer time to respond to wireless losses.

TCP Westwood

TCP Westwood [13, 17] implements a new congestion control algorithm that is based on end-to-end bandwidth estimation. In particular, TCP Westwood estimates the available bandwidth by counting and filtering the flow of returning ACKs and adaptively sets the *cwnd* and *ssthresh* after congestion by taking into account the estimated bandwidth. Upon receiving three duplicate acknowledgements the *cnwd* and *ssthresh* parameters are set according to the estimation while upon a timeout *cwnd* is set to one segment and *ssthresh* according to the estimated bandwidth. Hence, TCP Westwood ensures a faster recovery from losses that were not due to congestion, though the reduction to one segment upon a timeout remains the same as in the legacy implementations of TCP.

6.1.2 Link layer approaches

These approaches include methods that attempt to hide the lossy radio channel from the TCP sender. Forward error correction (FEC) and retransmission of lost packets in response to automatic repeat request (ARQ) deal with the problem of packet corruption locally and prevent congestion misinterpretation. The AIRMAIL and TULIP protocols, as presented in [18] and [19] respectively, lie in this category. Link layer protocols fit naturally to the layered structure of the network stack; a local problem, such a wireless loss, should be solved locally and therefore such approaches operate independently of higher layer protocols. These are the so-called TCP-unaware methods. One disadvantage of such approaches is the potentially adverse effects when interacting with higher-layer protocols [10], such as TCP. TCP-aware protocols encounter these adverse effects. In particular, Snoop [20] improves TCP performance by locally retransmitting corrupted packets, while at the same time it suppresses duplicate acknowledgements, so

as to avoid congestion avoidance routines at the TCP sender.

6.1.3 Split connections

Approaches lying in this category are based on the observation that since a TCP session operates in a mixed wired/wireless environment, merged through an access point, we could use a different protocol for each network part, i.e. we could split the TCP connection at the access point. Methods of this category concern exclusively wireless networks with access point infrastructure. According to these methods, the wired part runs the standard TCP protocol while the wireless part uses a specialized protocol tuned for wireless channels. Though, such approaches, like the Indirect-TCP [21] and the Mobile-TCP [22], violate the TCP semantics since the access point acknowledges TCP segments as soon it receives them and before they actually reach the destination host. Further, W-TCP [23] changes the timestamp field in the packet to account for the time spent idling at the base station.

6.2 Improving TCP performance in wireless networks with high delay variability

Several studies have been conducted in order to study high delay variability in wireless networks and its effects on TCP performance. Gurtov [24, 5] observes the presence of large delay variabilities in a GPRS wireless LAN and quantifies the negative impact of TCP spurious timeouts provoked by the delay variations. In [24], the author presents experiments using four different TCP implementations and their recovery capability against spurious timeouts and finally discusses several existing methods to alleviate the problem. Later, in [5] he studies the effects of delay spikes on TCP Tahoe, Reno, NewReno and Sack and recommends timing every TCP segment in order to achieve more conservative values for the sender's retransmission timer, so as to avoid spurious timeouts.

The above studies show that large delay variations are a reality for both GPRS and wireless LANs and have very serious impact on TCP protocol performance. Below we present three methods proposed in the bibliography that encounter the impact of large delay variabilities on TCP performance.

6.2.1 GBN retransmission policy

Leung *et al.* propose using a go-back-N retransmission policy at session layer instead of the typical selective repeat (SR) that TCP uses, in order to alleviate the negative effects of high delay variability. This method performs well in environments with strong temporal correlation, where if a packet is lost it is highly probable that the subsequent packets are lost as well. The GBN retransmission policy suggests retransmitting all outstanding packets upon a timeout or a loss detection of a specific packet and not only the last packet, as the typical TCP implementation does. This method can result in reduced throughput when packets are lost in a random fashion. In this case go-back-N retransmission wastes bandwidth. In the same paper, the authors propose another method to efficiently encounter large delay variability. In particular, they suggest selecting a larger retransmission timeout threshold by changing the RTO calculation formula. In this way, the retransmission timer of the sender becomes more robust against TCP spurious timeous. In the experiments conducted, the above two methods present throughput improvements up to 13.7% and 12% respectively. Note that, both proposed methods require changes to the TCP sender's implementation.

6.2.2 The Eifel algorithm

Ludwig and Katz [6] propose an enhancement to the TCP's error recovery scheme that helps the TCP sender alleviate the negative impact of spurious timeouts. The authors use the TCP timestamp option to eliminate the retransmission ambiguity caused by spurious timeouts and the corresponding retransmissions.

In order to eliminate the retransmission ambiguity we require extra information

so as to help the sender distinguish between a packet that was actually lost and a packet that was just delayed somewhere in the path. So, the algorithm having recorded the timestamp of a packets first transmission and of a potential retransmission, compares the recorded timestamp with the one of the corresponding ACK; if it is larger this indicates that the retransmission was spurious. Afterwards, the sender, having also recorded the *cwnd* and $ssthresh$ upon retransmission, restores the values of the *cwnd* and $ssthres$ and alleviates the unnecessary throttling. Drawbacks of this implementation are the bursty response of the sender upon spurious timeout detection. This inefficiency is encountered in [7], where the authors suggest a more smoothed restoration of the congestion control.

Although the Eifel algorithm performs very well in environments with spurious retransmissions (such as wireless environments) it has some disadvantages as well. Firstly, changes to the end hosts, i.e. the TCP senders, are required. Moreover the penalty of a single spurious retransmission can not be avoided in any case.

6.2.3 Delay jitter algorithm

Klein *et al* [3] examine how injecting delay to specific packets can improve end-to-end throughput. Delay injection can indirectly influence the sender's retransmission timer and therefore render it more robust. The authors propose three algorithms to inject delay: (i) Fixed Time - Fixed Delay (FTFD), where a fixed amount of delay is added over fixed periods of time, (ii) Random Time - Fixed Delay (RTFD), where predefined delay is injected of random intervals and (iii) Random Time - Random Delay (RTRD), where randomness is introduced in both the magnitude of the delay to inject and the frequency to be injected.

The performance evaluation of the above algorithms suggest throughput improvement by up to 8%. The sender avoids several TCP spurious timeouts, because of the robustness of his retransmission timer imposed by the delay injection. On the other hand, the proposed method is static and does not adapt to specific network characteristics; i.e. the optimal magnitude/frequency values do no perform well when the

propagation delay of the path, the number of mobile users or the packet error rate change.

6.2.4 Transparent TCP proxies

In [25], the authors present the design and implementation of a transparent TCP proxy that mitigates problems, such as poor loss recovery, excess queuing for long-lived flows, and link under-utilization for short-lived flows. The proposed mechanism does not require any changes to the TCP implementations in either mobile or fixed-wire end systems. The key features of the proposed scheme are (i) avoiding slow start during connection startup, (ii) error recovery is improved due to the extension of the SACK mechanism to the entire aggregate, (iii) the flow control mechanism algorithm manages proxy buffer space to ensure sufficient data is buffered to keep wireless link fully utilized, and (iv) scheduling of packets in the aggregate allows fair bandwidth allocation to flows, regardless of duration.

6.3 Relation to real traffic measurements

In Chapter 5, we saw that our adaptive delay injection method achieves significant performance improvements when the propagation delay at the wired part was fairly large, i.e. $> 15\,ms$ in single flow experiments. The authors in [26], study specific characteristics of TCP connections in the mixed wired and wireless LAN of the University of North Carolina at Chapel Hill (UNC) with almost 600 access points.

Among other things, the authors in [26] present measurements as far as the one-side transit time (OSTT) is concerned, defined as the the time spent on the LAN part (LAN OSTT) or the time spent on the WAN part (WAN OSTT) of the network. The cumulative probability for the average WAN OSTT shows that only 30% of the long wireless TCP connections (connections with more than 100 packets) have a WAN OSTT below 20 ms. This indicates that most wireless TCP connections tend to have fairly large delays, as suggested in the experiments we conducted.

Moreover, the authors shows the variability in delay presented in the LAN and WAN side of the TCP connections. The results show that the delay jitter in LAN side of the TCP connection is substantially larger than the one presented in WAN side. More specifically, they show that 70% of the wireless connections have a standard deviation below 20 ms. The delay to be injected according to our adaptive delay injection algorithm is strongly dependent of the average RTT of the path. Recalling that we capture delay variabilities using the delay spike identification mechanism, and that the results in [26] show that the WAN delay jitter is not large enough, we conclude that the calculation of the difference of *pdelay* minus *avdelay* in our algorithm, indeed, reflects the time spent on local retransmissions, since the delay variability in the wired part of the network is not large.

Still, there is a number of TCP connections that have fairly large standard deviation (10% of the wireless TCP connections pose a standard deviation above 100 ms), denoting that the agile adaptation of the average delay we propose may sometimes not be a robust metric.

Chapter 7

Conclusions and issues for further research

In this thesis we presented a method that improves TCP performance in wireless environments with bursty packet errors. Our work was motivated by the fact existing network protocol, originally designed for wireline network, present reduced performance when interacting with emerging wireless protocols. We therefore study the interaction of the most popular network protocol (TCP) with the most popular wireless protocol (802.11). The most important feature is that packet losses at the error-prone wireless channel as misinterpreted as signals of congestion by the TCP sender that unnecessarily reduces its transmission rate.

Our methodology suggests applying a strong data-link error recovery mechanism, so to prevent packet losses at the radio channel. Next, high delay variabilities, introduced by local retransmissions, are eliminated by artificial delay injection to the ACK packets of the TCP session. This way, we achieve significant improvements in the end-to-end throughput and the useful transmission ratio, especially in high delay paths. We conducted experiments in both infrastructure and multihop wireless environments in topologies with different network characteristics such as propagation delays, number of participating mobile nodes and packet error rates.

In this work, we show that our method adapts to networks with different path characteristics, in contrast to other methods proposed in the bibliography, rendering our method applicable in a wide range of network topologies. Moreover, our adaptive delay injection algorithm does require changes to the existing TCP protocol implementation and is implemented solely at the access point; hence, no changes to the end systems are required.

Part of our future work is to conduct experiments considering downlink data traffic with the aid of the modified round robin queuing mechanism, proposed in Chapter 4. Moreover, it would be interesting to implement a dynamic selection of the *RetryLimit* parameter of 802.11 that would adapt to changing channel quality and network characteristics. Recall that in Chapter 5 we showed that a less strong local error recovery mechanism can sometimes be more beneficial. Finally, the application of the proposed method to a real wireless network would be of great interest.

Bibliography

[1] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the sprint IP backbone," *IEEE Network*, vol. 17, No 6, pp. 6–15, Novemeber 2003.

[2] C. Williamson, "Internet Traffic Measurement," *IEEE Internet Computing*, vol. 5, No 6, pp. 70–74, November 2001.

[3] T. Klein, K. Leung, R. Parkinson, and L. Samuel, "Avoiding Spurious TCP Timeouts in Wireless Networks by Delay Injection," *in Proceedings of IEEE GLOBECOM*, Novemeber 2004.

[4] K. Leung, T. Klein, C. Mooney, and M. Haner, "Methods to Improve TCP Throughput in Wireless Networks With High Delay Variability," *in Proccedings of IEEE Veh. Tech. Conf.*, 2004.

[5] A. Gurtov, "Making TCP Robust Against Delay Spikes," *University of Helsinki, Department of Computer Science, Technical Report C-2001-53*, November 2001.

[6] R. Ludwig and R. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions," *in ACM Computer Communication Review*, vol. 30, No 1, January 2000.

[7] A. Gurtov and R. Ludwig, "Responding to Spurious Timeouts in TCP," *in Proceedings of IEEE INFOCOM*, March 2003.

[8] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," *RFC2988*, 2000.

[9] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," *RFC1323*, 1992.

[10] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Transactions on Networking*, pp. 5 (6):756–769, December 1997.

[11] Network Simulator 2, availiable at http://www.isi.edu/nsnam/ns/

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," *RFC 2018*, April 1996.

[13] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, "TCP Westwood: Congestion Window Control Using Bandwidth Estimation," *in Proceedings of IEEE Globecom 2001*, November 25-29, 2001.

[14] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, April 1999.

[15] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, vol. 25, N. 5, pp. 10–23, November 25-29, 2001.

[16] F. Peng, S. Cheng, and J. Ma, "A Proposal to Apply ECN to Wireless and Mobile Networks," *in Proceedings of INET '00*, July 2000.

[17] R. W. A. Z. C. C. S. M. M. Gerla, M. Y. Sanadidi, "TCP Westwood: Congestion Window Control Using Bandwidth Estimation," *in Proceedings of IEEE Globecom 2001*, vol. 3, pp. 1698–1702, November 25-29, 2001.

[18] E. Ayanoglu, S. Paul, T. F. LaPorta, K. Sabnani, and R. Gitlin, "AIRMAIL: A Link-Layer Protocol for Wireless Networks," *ACM ACM/Baltzer Wireless Networks Journal*, pp. 47–60, February 1995.

[19] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP Performance over Wireless Networks at the Link Layer," *in Mobile Networks and Applications*, vol. 5, N. 1, pp. 57–71, 2000.

[20] H. Balakrishnan, "Improving TCP/IP Performance over Wireless Networks," *in Proceedings of ACM MOBICOM '95*, 1995.

[21] B. Badrinath and A. Bakre, "I-TCP: Indirect TCP for Mobile Hosts," *in Proceedings of 15th International Conference on Distributed Computing*, May 1995.

[22] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," *in Proceedings of SIGCOMM '97*, October 1997.

[23] K. Ratnam and I. Matta, "WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links," *in Proceedings of 3rd IEEE Symposium on Computers and Communications (ISCC '98), Athens, Greece*, 1998.

[24] A. Gurtov, "Effect of Delays on TCP Performance," *in Proceedings of IFIP Personal Wireless Communications*, August 2001.

[25] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt, "Flow Aggregation for Enhanced TCP over Wide-Area Wireless," *Proceedings of the IEEE INFOCOM 2003*, March 29-April 4, San Francisco, USA, 2003.

[26] F. Hernández-Campos and M. Papadopouli, "Assessing The Real Impact of 802.11 WLANs: A Large-Scale Comparison of Wired and Wireless Traffic," *14th IEEE Workshop on Local and Metropolitan Area Networks*, Chania, Crete, Greece, September 18-21, 2005.

[27] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt, "Performance optimizations for wireless wide-area networks: comparative study and experimental evaluation," *Proceedings of the 10th annual International Conference on Mobile Computing and Networking*, Philadelphia, PA, USA, 2004.