Department of Computer Science
Series of Publications A
Report A-2010-6

Cross-layer Assisted TCP Algorithms for Vertical Handoff

Laila Daniel

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, University Main Building, on 20th November 2010, at ten o'clock.

University of Helsinki
Finland

Contact information

Postal address:
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: postmaster@cs.helsinki.fi (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 51120

ISSN 1238-8645
ISBN 978-952-10-6660-3 (paperback)
ISBN 978-952-10-6661-0 (PDF)
Computing Reviews (1998) Classification: C.2.6, C.4
Helsinki 2010
Helsinki University Print

Cross-layer Assisted TCP Algorithms for Vertical Handoff

Laila Daniel

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
laila.daniel@cs.helsinki.fi
http://www.cs.helsinki.fi/laila.daniel

PhD Thesis, Series of Publications A, Report A-2010-6
Helsinki, November 2010, xiv+84+72 pages
ISSN 1238-8645
ISBN 978-952-10-6660-3 (paperback)
ISBN 978-952-10-6661-0 (PDF)

Abstract

The ever expanding growth of the wireless access to the Internet in recent years has led to the proliferation of wireless and mobile devices to connect to the Internet. This has created the possibility of mobile devices equipped with multiple radio interfaces to connect to the Internet using any of several wireless access network technologies such as GPRS, WLAN and WiMAX in order to get the connectivity best suited for the application. These access networks are highly heterogeneous and they vary widely in their characteristics such as bandwidth, propagation delay and geographical coverage. The mechanism by which a mobile device switches between these access networks during an ongoing connection is referred to as vertical handoff and it often results in an abrupt and significant change in the access link characteristics. The most common Internet applications such as Web browsing and e-mail make use of the Transmission Control Protocol (TCP) as their transport protocol and the behaviour of TCP depends on the end-to-end path characteristics such as bandwidth and round-trip time (RTT). As the wireless access link is most likely the bottleneck of a TCP end-to-end path, the abrupt changes in the link characteristics due to a vertical handoff may affect TCP behaviour adversely degrading the performance of the application.

The focus of this thesis is to study the effect of a vertical handoff on TCP behaviour and to propose algorithms that improve the handoff behaviour of TCP using cross-layer information about the changes in the access link

characteristics. We begin this study by identifying the various problems of TCP due to a vertical handoff based on extensive simulation experiments. We use this study as a basis to develop cross-layer assisted TCP algorithms in handoff scenarios involving GPRS and WLAN access networks. We then extend the scope of the study by developing cross-layer assisted TCP algorithms in a broader context applicable to a wide range of bandwidth and delay changes during a handoff. And finally, the algorithms developed here are shown to be easily extendable to the multiple-TCP flow scenario. We evaluate the proposed algorithms by comparison with standard TCP (TCP SACK) and show that the proposed algorithms are effective in improving TCP behavior in vertical handoff involving a wide range of bandwidth and delay of the access networks. Our algorithms are easy to implement in real systems and they involve modifications to the TCP sender algorithm only. The proposed algorithms are conservative in nature and they do not adversely affect the performance of TCP in the absence of cross-layer information.

Computing Reviews (1998) Categories and Subject Descriptors:
C.2.6 Computer-Communication Networks/ Internetworking
C.4 Performance of Systems

General Terms:
TCP, Vertical Handoff, Wireless Access Networks, Wireless Internet

Additional Key Words and Phrases:
Cross-layer Notifications, Performance Analysis

Acknowledgements

After playing football with his brother in our yard, my eight-year old son said to me, "mother, in a game, even if you touch the ball with your foot it is a great achievement". My thinking at this moment is similar to his, having an opportunity to learn about the exciting world of Internet, witnessing the great communication revolution happening around us and being able to participate in it, and working with exceptional and dedicated people, I consider myself very fortunate. I thank every one who has been with me all along in this journey with a heart overflowing with gratitude.

I would like to express my deep gratitude to my supervisors Professor Kimmo Raatikainen, Lecturer Markku Kojo and Professor Jussi Kangasharju for guiding my research and giving me invaluable support at every step on my way. Kimmo taught several courses that helped me to get a broad understanding of the field of mobile and wireless networking and also gave me much encouragement in my research work. He gave valuable comments on the initial work in this thesis and it is very unfortunate that I could not share the completion of my thesis with him. My gratitude to Markku is ineffable, but for him there will not be this thesis. The countless discussions I had with Markku on a day-to-day basis contributed much to my understanding of my research topic. Markku also meticulously read several drafts of the thesis and gave numerous suggestions for improving it. I am grateful to Jussi for his valuable advice in completing this work and taking care of all the official matters.

I thank Professor Jarmo Harju and Professor Jukka Manner, the pre-examiners of my thesis, for their suggestions and comments on my thesis which helped in improving it. My thanks to Marina Kurtén for correcting the language of this thesis, though any remaining errors are mine alone. I am grateful to Anu Hirsiaho and Markku Kojo for helping me to prepare the popular abstract for this thesis in Finnish.

I am grateful to both the current and the former members of our project team, Aaron Yi Ding, Aki Nyrhinen, Davide Astuti, Ilpo Järvinen, Krishnan Narayanan, Lauri Hyttinen and Seppo Hätönen for creating a cordial

and pleasant working environment. Special thanks to Ilpo for being a co-author of one of the papers in this thesis and for providing many evaluation scripts and to Aki for many lively discussions. I would also like to thank Pasi Sarolahti and Jouni Korhonen for many discussions and for the joint work which resulted in a publication. My thanks to Matti Luukkainen, with whom I wrote an early paper, for showing me the skills useful in writing papers. I would like to thank everyone in the Nodes group for the enjoyable presentations and lively get-togethers. I would like to acknowledge the financial support received from the MERCoNe, WISEciti and ICT SHOK projects and I also wish to express my thanks to the partners of the projects for the fruitful discussions and for the joint activities.

It is a pleasure to thank Tiina Niklander for the help and advice she readily gave me whenever I needed. Her enthusiasm and friendliness lifted up my spirits on numerous occasions when I despaired about the progress of my research work. Tiina also gave me valuable comments which helped in organizing the thesis. I would like to thank Dr. Timo Alanko for helping me to get a research position here. I am grateful to Satu Eloranta for her kindness, friendship and also for bringing a green touch to my office. My thanks to my former officemate Oriana Riva for the laughter and fun we shared which brought light moments during many long days spent in office. I would like to thank Janey Hu, Ramya. S, Taneli Vähäkangas and Tancred Lindholm for the interesting conversations we had. My thanks to Pasi Rastas for the latex templates for this thesis and to Jussi Kollin for pointing out the missing 'maketitle' in my document.

I thank the Department of Computer Science, University of Helsinki for providing me an excellent working environment. It is an honour for me to do my research in this esteemed institute. I would like to thank Professors Esko Ukkonen, Hannu Toivonen and Jukka Paakki, the current and the former Heads of the Department for giving me this opportunity. My thanks to Dr. Greger Lindén, secretary for PhD studies, for all the advice and help he gave me in the official 'graduating' process. My sincere thanks to the system administrators of the Computer Science Department especially Petri Kutvonen, Pekka Niklander and Ville Hautakangas for help with the computing facilities. I would like to thank the office staff of the Department for their help, especially Päivi Karimäki-Suvanto, our former office manager and Pirjo Kokkonen, our department secretary.

My sincere thanks to Professor Nahid Shahmehri, Linköping University and to the Swedish Institute for opening the doors to my Scandinavian experience. Thank you Nahid for teaching me to say 'yes' to life.

My heartfelt thanks to all my teachers especially those from the elementary school who gave me a love of learning. Thanks to all my classmates for making my student life a memorable one.

I am deeply indebted to Docent Jukka Rajantie, pediatric hematologist, not only for being a physician to my son but also the kind interest he showed in the progress of my research work. Even in my difficult moments during hospital visits, he kindled a faint hope that one day I could graduate.

I express my deep gratitude to the children, parents and teachers of Västersundom skola, for giving my children a joyous childhood which I sometimes forgot in the midst of my studies. My thanks to the personnel in Lilla My daycare who helped us to start our day with a bright smile.

It is the strength of prayer that sustained me and made my research joyous and fruitful. My gratitude to Rev. Jukka Yrjölä for giving me spiritual strength. Our prayer group led by Hiljä Gröhn is a place of comfort and joy that made me feel at home. I believe that your love and prayers helped me to go forward in difficult moments.

I owe much to my friends, Anu Hirsiaho, Eija Helminen, Erja Airaksinen, Heli Astala, Heljä Ylönen, Katja Hintsanen, Katri Luomaranta, Kylli Lappi, Marjo and Markko Raina, Mari Hannuksela, Maria Erkkilä, Maria and Tapio Karvonen, Mei Aiping, Merja Vuori, Ping Gu, Riitta Bruci, Sirkka Niemi, Terhi Rinkinen, and Tiina Heinonen. I asked them a million favours and they obliged with infinite smiles. Thanks for your love, support and prayers.

I am deeply indebted to Eija Helminen, my friend, who knows all the minute steps in my thesis process. Thanks Eija for our 'mobile' prayers. I cannot describe my surprise and joy when I saw Heljä just before my presentation in a Tenerife conference. Thanks Mei for being my spiritual sister. Thanks to Raina family for the wonderful times we have shared.

I am grateful to Viji and her mother, Ingrid Wanhammar, Jakob Gustafsson and family, Anna Martinsson and family and Annikka Pohl and family for the sweet, memorable time we had in Linköping. Thanks to Annikka and family for our annual meetings in either Helsinki or Stockholm. I wish to thank Ajith Thayil, P. Ajith, Nidhi Gupta, Dallys Sebastian for helping us at the time of our arrival in Finland.

I thank Dr. A. K. Kasthurba and Dr. V. K. Govindan for their support and understanding in my difficult moments. I would like to thank Dr. Lillikutty Jacob, Dr. Paul Joseph, Dr. V. J. Kurian, Dr. M.P Chandrasekharan, Dr. Elizabeth Elias, Dr. M. N. Neelakantan, Dr. K. J. John, Dr. Priya Chandran, Ms. Ancy Jose, Dr. M.G. Sreekumar, Mr. N. Rajendrakumar, Mr. E. Balakrishnan for all their help and best wishes.

I am deeply indebted to Dr. Liginlal and family who have been a source of immense support, help, generosity and kindness all through.

I am grateful to Chellu, Cini, Mini, Shabana, Lancy, Usha, Maari and Radhika for their love and affection towards me. Padma miss, you always were a source of love and encouragement.

I am thankful to S. Radhika, Vivek, Zarina, Swapna, Venu, Smitha, Vinod, Ranju, Shyam, Mahadeven, Sanil, Subramani, Manoj, Srikant, Prasant, Renjish, Lisa and John for their warm friendship. Special thanks to Radhika and Vivek for their generous help in many occasions.

I have no words to thank my brothers Shibu and family, Biju and family, my sister-in-law Vaidehi and family, Vijayaraghavan and family, Kunjay and family, James achachan and family, Abraham Master and family, Achamma Teacher and family, Raju and family, Achanmonachayan and family, Jolly and family, Adoorappachan and family, Sosamma teacher and family, Rev. V. Jose, Kavitha and family, Thamaravelil family and Puthuvilayil family for their love, support and prayers.

Shibu, Sheeba, Biju, Saritha, Vaidehi and Jyothi, your faith in me has been a source of immense strength. I am very grateful to Appa, my father-in-law, for his love and blessings.

This thesis has been a long journey as we progressed in our bedtime reading from Goodnight Moon to Matilda. These moments with my children, Aravind and Nirmal, filled me with joy, optimism and hope. Paapu and Nimmu, you gave me much love, support and understanding which was so important for me during this work. I am grateful to my husband, Krishnan, for enriching my life and for supporting me throughout this thesis process.

My parents were also my first teachers and what I owe them I cannot describe in words. Their love and prayers have always been a source of strength to me. I dedicate this thesis to them.

I am sure that I have forgotten to mention some whom I should have thanked individually. My apologies and collective thanks to them.

I praise and thank the Almighty God for bringing me from a small village in Kerala, India to Finland and allowing me to write these lines.

Helsinki, 27.10.2010
Laila Daniel

List of Original Publications

This doctoral thesis is based on the following four research papers, which are referred to in this thesis as Papers 1-4. These papers are included at the end of this thesis.

Paper 1: Laila Daniel and Markku Kojo.
Adapting TCP for Vertical Handoffs in Wireless Networks, *Proceedings of the 31st IEEE Conference on Local Computer Networks, (LCN'06), November 2006 pages 151-158.*

Paper 2: Laila Daniel and Markku Kojo.
Employing Cross-layer Assisted TCP Algorithms to Improve TCP Performance with Vertical Handoffs, *International Journal of Communication Networks and Distributed Systems (IJCNDS), Volume 1, Issue 4/5/6, November 2008, pages 433-465.*

Paper 3: Laila Daniel, Ilpo Järvinen and Markku Kojo.
Combating Packet Reordering in Vertical Handoff Using Cross-Layer Notifications to TCP, *Proceedings of the IEEE Conference on Wireless and Mobile Computing (WiMob'08) October 2008, pages 297-303.*

Paper 4: Laila Daniel and Markku Kojo.
The Performance of Multiple TCP Flows with Vertical Handoff using Cross-Layer Notifications to TCP, *Proceedings of the 7th ACM International Symposium on Mobility management and Wireless Access, (MobiWac'09)*, October 2009, pages 17-25.

List of Abbreviations

$2G/3G$	Second/ Third Generation Cellular Technology
$3GPP$	Third Generation Partnership Project
$802.xx$	Wireless LAN Standards developed by IEEE
ACK	Acknowledgment
AP	Access Point
AR	Access Router
BBM	Break-before-make
BDP	Bandwidth Delay Product
BS	Base Station
BSC	Base Station Controller
BSS	Base Station Subsystem
BTS	Base Transceiver Stations
BU	Binding Update
$CDMA$	Code Division Multiple Access
CoA	Care-of Address
DS	Distribution System
$EDGE$	Enhanced Data Rates for GSM Evolution
ESS	Extended Service Set
$ETSI$	European Telecommunications Standards Institute
FA	Foreign Agent
FTP	File Transfer Protocol
$GGSN$	Gateway GPRS Support Node
$GMSC$	Gateway MSC
$GPRS$	General Packet Radio Service
GSM	Global System for Mobile Communications
HA	Home Agent
HIP	Host Identity Protocol
$HSDPA$	High-Speed Downlink Packet Access
$HSPA$	High-Speed Packet Access
$HSUPA$	High-Speed Uplink Packet Access
$IEEE$	Institute of Electrical and Electronics Engineers

$IETF$	Internet Engineering Task Force
IP	Internet Protocol
$IPv4/IPv6$	IP Version 4/ IP Version 6
$ISDN$	Integrated Services Digital Network
ISO	International Organization for Standardization
LAN	Local Area Network
LLC	Logical Link Control
LTE	Long Term Evolution
MAC	Medium Access Control
MBB	Make-before-break
$MBWA$	Mobile Broadband Wireless Access (Mobile-Fi)
MIP	Mobile IP
$MIPv4/MIPv6$	MIP Version 4, MIP Version 6
MN	Mobile Node
$MPTCP$	Multipath TCP
MSC	Mobile Switching Center
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NGN	Next Generation Networks
NIC	Network Interface Card
$PSTN$	Public Switched Telephone Network
RAN	Radio Access Network
RFC	Request For Comments
RNC	Radio Network Controller
RTO	Retransmission Timeout
RTT	Round-Trip Time
$SCTP$	Stream Control Transmission Protocol
SDR	Software Defined Radio
$SGSN$	Serving GPRS Support Node
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UE	User Equipment
$UMTS$	Universal Mobile Telecommunications System
$UTRAN$	UMTS Terrestrial Radio Access Network
WAN	Wide Area Network
$WCDMA$	Wideband CDMA
$WiFi$	Wireless Fidelity
$WiMAX$	Worldwide Interoperability for Microwave Access
$WLAN$	Wireless LAN

Contents

Chapter 1

Introduction

In this chapter we introduce the problem examined in this thesis along with a brief overview of the research area. The rise of the wireless access to the Internet has spawned a variety of wireless access technologies that enable a mobile device to connect to the Internet using multiple radio interfaces. The widely differing characteristics of these access technologies impact on the performance of transport protocols such as TCP used by most of the common Internet applications. In this thesis we examine how TCP is affected when a mobile device switches from one wireless access network to another and propose solutions to adapt TCP to this change by making use of cross-layer information about the characteristics of the access networks.

1.1 Future Heterogeneous Access Systems and Challenges

The enormous growth of the Internet and mobile telephony in the last decade or so has led to the exciting prospect of the interweaving of the two technologies leading to the emergence of the *Wireless Internet* [86,94,121]. This scenario has been anticipated in the prescient papers of Mark Weiser on Ubiquitous Computing [150] and Kleinrock on Nomadic Computing [89]. Among the notable early works in this direction are [19, 20, 33, 90], which deal with various issues arising from incorporating mobile devices to the Internet. From its inception in the mid-90's there has been sustained and intense research in the area of Wireless Internet led by numerous experimental investigations coupled with rapid technological strides. In this emerging scenario it is expected that a wide range of mobile services and applications such as WWW, real-time multimedia, games, news feeds and social networking will be available to mobile users in a seamless way

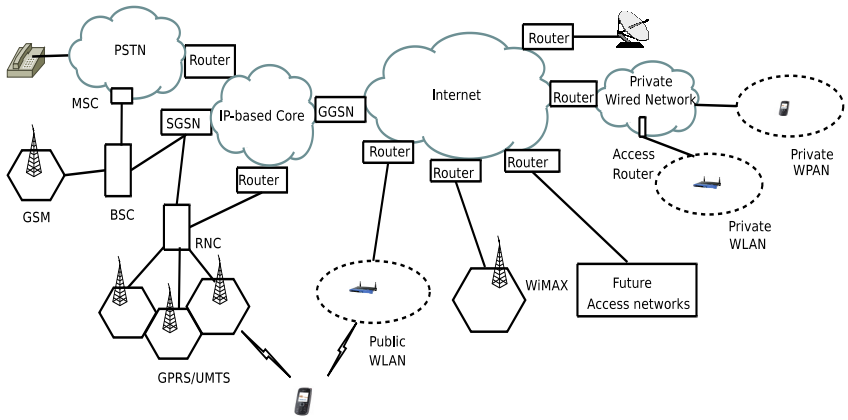


Figure 1.1: An Example Architecture of Future Heterogeneous Access Systems. Adapted from [141]

regardless of their geographical location. In this context, the services and applications will have the need to take into account the characteristics of the mobile environments to operate satisfactorily [121].

The current state of the evolution of the Wireless Internet is guided by the goals of ubiquitous computing [150, 151] in which the support for mobility, multimedia communications, wireless broadband, Quality of Service (QoS) guarantees, seamless coverage and security provide the basis for a multitude of services and applications that can be readily accessed by mobile users. The emergence of the concept of *next generation network* (NGN) provides the technological infrastructure necessary to support the above-mentioned communication goals [80]. An NGN is a collection of wired and wireless networks with the Internet Protocol (IP)-based backbone (core). NGN is also known as *all-IP network* as IP glues all these networks together. The key features of the NGN include the anytime anywhere connectivity, support for data and multimedia services, use of devices which can support several access technologies and support for integrated service access from various service providers.

Currently there are a wide variety of wireless access networks such as Wireless Personal Networks (WPN) (e.g., Bluetooth), Wireless Local Area Networks (WLAN) (e.g., IEEE 802.11 a/b/g/n), and Wireless Wide Area Networks (e.g., GPRS, UMTS, WiMAX, Satellite networks) that a mobile device can use to connect to the Internet. Figure 1.1 illustrates an example architecture of future heterogeneous access networks. The inter-networking of the various wireless networks is highly desirable to provide ubiquitous connectivity as no single wireless network is capable

of providing the best connectivity, high bandwidth and wide geographical coverage anytime anywhere. The heterogeneity of these access networks arising from their diverse underlying radio technologies, geographical coverage ranging from a small area to a wide area, bandwidth ranging from a few Kbps to several Mbps, latency variations by an order of magnitude or more and operation under possibly different administrative domains poses a challenge to the inter-networking of the different wireless networks.

The mobile devices should support access to the different wireless access technologies so that they can seamlessly roam among the different networks to provide anytime anywhere connectivity. These devices, known as *multimodal* devices, are currently of two types, namely, *multimode terminals* and *software-defined radios* (SDRs) [99]. A multimode terminal has several network interface cards (NICs) and software that enables switching between the network interfaces to connect to the different access networks. The multimode terminals should activate the different network interfaces to detect the service advertisements from different access networks. As battery power is a critical resource for a mobile terminal, keeping all the network interfaces alive is not a viable option as it consumes considerable power even in the 'idle' mode when no packets are either being received or sent. So a tradeoff exists between the power efficiency and network discovery time which has to be minimized. SDRs use adaptable software to implement radio functions that operate on a generic, flexible hardware platform consisting of digital signal processors and general purpose microprocessors. SDRs allow reconfiguration of a wide range of access technologies such as WLAN, GPRS and WCDMA by selecting the appropriate software modules. Though this is an attractive technology, advances in reducing the power consumption and the processing time along with cost reduction are needed for a wide-spread adoption of SDR [141].

Based on user preferences and service requirements, a mobile device should be able to detect and select the best available access network at a given time and changeover to the selected network to maintain connectivity to the Internet. This switching process from one access network to another is referred to as *handoff* [1] [100]. Handoff management should allow a mobile device to roam among multiple wireless networks in a manner transparent to applications with minimal disruption in connectivity. The development of handoff-decision algorithms is an active research area [111]. When a mobile device moves from one access network to another due to a handoff, the change in the IP address of the mobile device creates problems as both routing and host identity in the Internet are based on the IP address.

[1] The terms "handoff" and "handover" are used interchangeably in the literature.

Mobility issues such as the management of mobility and handoff, the choice of the layer best suited for implementing mobility management and the reduction in the handoff latency are key areas of research [46, 91, 119].

Even in the case where the mobility and handoff mechanisms provide a smooth handoff, the transport protocols may be affected by the handoff as the protocol behaviour depends on the end-to-end path characteristics which may change significantly due to a handoff. As the Transmission Control Protocol (TCP) is the dominant transport protocol in the Internet, it is important to study how TCP is affected by a handoff and devise solutions to improve the performance of TCP in a handoff. This topic has been an active area of research [59, 64, 67, 87, 88, 124, 125, 133, 137, 138] and is also the focus of the study presented in this thesis.

Provisioning of QoS in terms of guaranteeing a certain bandwidth, throughput, delay variation bound and cost in a heterogeneous mobile environment is a challenge as the different wireless access networks that a mobile device uses to connect to the Internet have widely different characteristics. The issue of secure data flow is also a major concern in heterogeneous access systems. As the various networks that participate in an NGN may belong to different network operators and may operate under different administrative domains, it is a challenge to develop suitable billing, accounting and authentication mechanisms that satisfy the end user. A discussion of the various research issues in the design and deployment of future heterogeneous access networks can be found in [141].

1.2 Motivation for Our Research

Contemporary mobile devices increasingly support multiple radio technologies such as WLAN, GPRS, UMTS, HSPA and WiMAX to provide wireless access to the Internet. For instance, the current state-of-the-art mobile phone, Nokia N8-00, supports multiple radio interfaces such as GPRS/EDGE, HSDPA (maximum speed up to 10.2 Mbps), HSUPA (2.0 Mbps) and WLAN IEEE 802.11 b/g/n along with key applications such as Calendar, Contacts, music player, Internet, messaging, photos, Ovi Store that supports games, maps and videos, Web TV, Office document viewers, Video & photo editor, Mail and Radio [108]. It is envisaged that in the future mobile devices can support even ten or more radio interfaces to support a wide range of Internet applications using a broad range of wireless access technologies [66, 126].

The transport protocol TCP [118, 146, 155] is used by a wide range of applications in the Internet ranging from the classical ones such as e-mail,

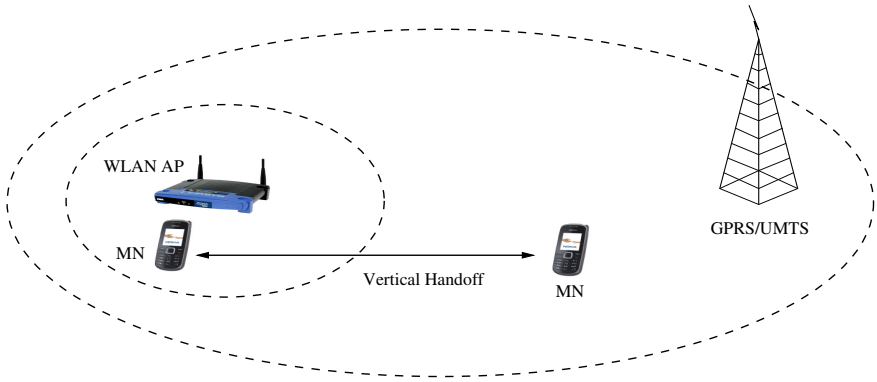


Figure 1.2: Vertical Handoff

Web browsing and file-transfer to the more recent ones such as streaming audio, video and P2P [37, 128, 144, 148]. Recent measurement studies on Internet traffic show that TCP still dominates the *volume* of the Internet traffic (in terms of packets and bytes) whereas UDP now often accounts for the larger fraction of the total *number* of flows on a given link due to the rise in the streaming and P2P traffic [34,35,158]. TCP is an *end-to-end protocol* as it is implemented both at the sender and the receiver and the network has no 'state' information about the communicating end points using the TCP protocol. This characteristic of TCP is vital to the scalability of the network, robustness of the network operation in the case of the failure of the network nodes and ease of deployment of new applications. TCP provides the twin functions of *reliable data transfer* and *congestion control* which enable the applications to adapt their data rates to the available capacity in the network. The behaviour of TCP is highly dependent on the end-to-end path properties of the connection such as the round-trip time (RTT) and the bottleneck bandwidth. As a wireless access network connects a mobile device to the Internet at the edge of a TCP connection, it strongly influences the TCP behaviour in often being the bottleneck of the end-to-end path [45, 93].

The term *vertical handoff* refers to the changeover in which an active mobile node changes an ongoing connection from one wireless access network to another which uses a different link technology [100, 145]. For example, a mobile device equipped with network interface cards to connect to WLAN and GPRS may use a WLAN interface to connect to the Internet when it is indoors and switch over to the GPRS network when the mobile device moves out of the range of the WLAN. Figure 1.2 shows a vertical handoff scenario between WLAN and GPRS networks. As a

vertical handoff results in abrupt significant changes in the bottleneck link characteristics, it may take TCP several round-trip times (RTT) to adapt to the path changes during which TCP may experience problems such as false congestion signals, packet reordering, packet losses and unnecessary retransmissions which affect the application performance.

Mobility protocols such as Mobile IP [115] aim at hiding the host mobility from the layers above IP. While this is highly desirable from the modularity perspective, it is not a viable approach if the implications of mobility affect the segment delivery at the transport layer. A vertical handoff that causes a significant change in the access link characteristics cannot be totally hidden from the transport layer even if the handoff latency is reduced to a minimum and no packet drops occur. Regardless of how smooth or seamless the mobility protocols are made out to be, the problems of TCP due to a vertical handoff cannot be wished away and deserve to be examined to find appropriate solutions. This observation is based on numerous studies in this area including our own work [36, 41, 64, 88].

A natural approach to the solution of the problem of TCP behaviour in a vertical handoff is to provide information about the changes in the access link characteristics to TCP to enable it to adapt sooner to the new end-to-end path after a handoff. Without this information, TCP on its own can of course adapt to the new path, but the adaptation may take several round-trip times and during this interval TCP may suffer from unnecessary retransmissions, packet losses and timeouts. The term *cross-layer information* is used in this context to refer to the information given to TCP from the lower link layer about the changes in the access link characteristics due to a handoff. However as the gains of a cross-layer approach are often at the price of modularity, the cross-layer design has to be done carefully to avoid introducing unintended coupling effects between the layers [85]. So the cross-layer assisted TCP algorithms have to be designed conservatively to avoid aggressive TCP behaviour [130]. The cross-layer assisted TCP algorithms that are presented in this thesis have been designed with this consideration in view.

Figure 1.3 positions our research work in the context of the research challenges in the future heterogeneous access systems.

Our research described in the thesis presents a systematic study of the behaviour of TCP in a vertical handoff and uses this study as a basis to devise solutions to overcome the problems of TCP in handoff. A key enabler of our solutions is the cross-layer information from the link layer about the occurrence of handoff along with a rough estimate of the bandwidth and delay characteristics of the access links involved in the handoff. We build

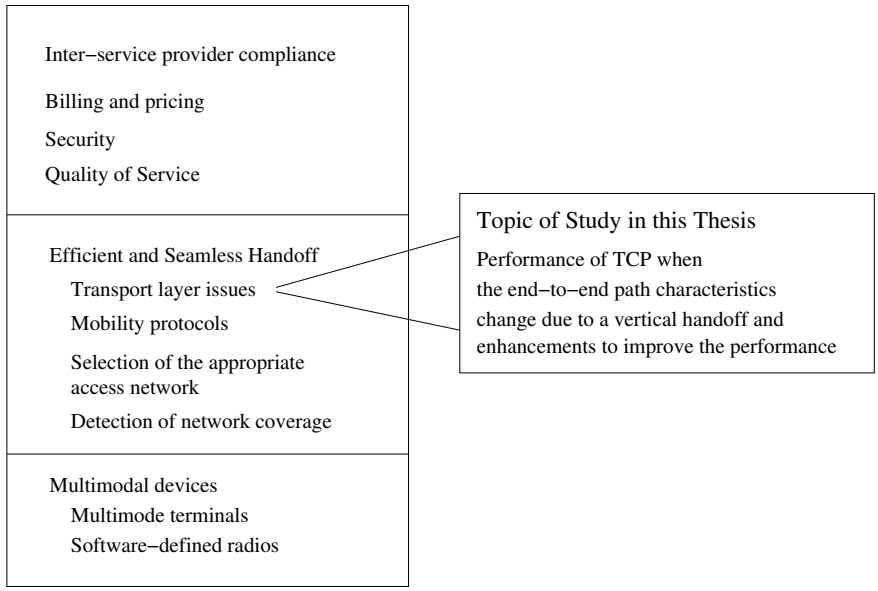


Figure 1.3: Research Challenges in Future Heterogeneous Access Networks. Adapted from [141]

a simulation model for the vertical handoff that is implemented in the network simulator ns-2 [106]. This model is used to study the behaviour of TCP when a handoff occurs at various instances of the different phases of a TCP connection such as slow start, congestion avoidance, fast recovery etc. Simulation provides a compact, versatile tool to examine a wide range of scenarios quickly which is essential in studying the general behaviour of a complex protocol such as TCP [114].

1.3 Contributions

The main contributions of this thesis are described below:

- This thesis presents *a thorough study of the behaviour of TCP in various handoff scenarios leading to the identification of the different problems of TCP in a handoff.* Our study not only includes the handoff between the most common wireless access networks of today but also handoff involving access links of arbitrary bandwidth and delay. This helps us to find the sensitivity of the TCP performance on the changes in the access link's bandwidth and delay due to a vertical handoff.

- *Based on the study of TCP behaviour in a vertical handoff we develop robust, conservative, cross-layer assisted TCP algorithms to improve the vertical handoff performance of TCP.* These algorithms can be implemented at the TCP sender and they do not require any modifications to the TCP receiver. Our algorithms are designed to be conservative in nature in that they do not make TCP respond aggressively to the cross-layer information. Our algorithms carefully combine the cross-layer information with the current state of TCP in deciding the appropriate actions. The solutions proposed here to improve TCP in a vertical handoff are fairly robust in that they can tolerate modelling errors such as inaccuracies in the cross-layer information given to TCP. Besides, the proposed solutions are easy to implement in real systems as we found from our experiments in implementing these algorithms in Linux kernel version 2.6.18 [83].

The research reported in this thesis has been published in four original publications that are referred to as Papers 1, 2, 3 and 4 in this thesis. The specific contributions of the original papers are described below.

Paper 1: This paper is based on a study of TCP behaviour in vertical handoff in the specific setting of GPRS and WLAN where an order of magnitude change in bandwidth and delay of the access links can occur due to a handoff. We explore the problems of TCP due to a handoff and propose enhancements to the TCP sender algorithm that make use of the binary information given to the TCP sender on whether a significant increase or decrease (i.e., an order of magnitude change) in the access link bandwidth and/or delay due to a handoff has occurred. This study clearly brings out the beneficial role of cross-layer information regarding handoff in improving TCP behaviour even when this information is very limited. A novel simulation model designed to reflect vertical handoff more realistically by changing the routes the packets take before and after a handoff (rather than modelling a vertical handoff by changing the link characteristics) is presented in this paper and this model is used in all our subsequent work.

The work in this paper was carried out in collaboration with Markku Kojo. Both authors contributed equally to the design of the algorithms and the planning of the experiments. The ns-2 implementation for the simulation, the simulation and analysis of the experiments were carried out by the author with the help of the coauthor. The paper was written mainly by the author with valuable suggestions from the coauthor. The simulation model used in this study is adapted from the model developed by Pasi Sarolahti, Markku Kojo and the author in the paper [133].

Paper 2: This journal paper is an extension of the work published in two workshop papers [41, 42]. In this work, we first present a systematic study of TCP behaviour in a vertical handoff under a wide range of bandwidth and delay values of the access links. This study has brought out the various problems of TCP in the different handoff scenarios. Although the problems of TCP such as the occurrence of spurious RTOs, packet reordering, packet losses, and unused connection time have been reported in the literature previously, our study clearly identifies the specific handoff scenarios in which these problems arise. Our study also points out an interesting result that even in the case of a handoff between access links of the same bandwidth-delay product (BDP), the packet losses can occur if the old link has much larger bandwidth than that of the new link and the packet losses can increase considerably as this ratio between the bandwidth of the links increases. Besides this study provides a useful basis for the design and evaluation of algorithms to improve TCP performance in a vertical handoff.

In the second part of this paper, the algorithms given in Paper 1 are refined to deal with additional information about the characteristics of the access links involved in a handoff. In our model a rough estimate of the bandwidth and delay of the access links involved in a handoff is given to the TCP sender along with the cross-layer notification about the occurrence of a handoff. Based on detailed simulation experiments, we show that the proposed algorithms are effective in avoiding spurious RTOs, in reducing packet losses, in improving the link utilization immediately after a disconnection and in converging quickly to the RTO value of the new end-to-end path. The problem of packet reordering is not addressed in this paper.

The work in this paper was done in collaboration with the coauthor Markku Kojo. Both authors contributed equally to the design of the algorithms and the planning of the experiments. The ns-2 implementation for the simulation, the simulation experiments and the analysis of the experiments were carried out by the author with useful suggestions from the coauthor. The paper was written mainly by the author with valuable feedback and suggestions from the coauthor.

Paper 3: In this paper we study the problem of packet reordering in TCP due to a vertical handoff and design an algorithm to mitigate the effects of packet reordering. The idea behind this algorithm is to determine how likely is packet reordering given the bandwidth of the two access links and use this information along with Duplicate Selective acknowledgements (DSACKs), an advanced feature of TCP, for detecting unnecessary retransmissions. Our algorithm enables TCP to utilize the high bandwidth that

may be available after a handoff.

The work in this paper was done in collaboration with Markku Kojo and Ilpo Järvinen. Ilpo Järvinen participated in discussions on the problem. Markku Kojo and the author developed the algorithms while the author carried out the ns-2 implementations, experiments and the analysis of the results with useful suggestions from Markku Kojo. This paper was written by the author with corrections and suggestions from the coauthors.

Paper 4: In this paper we study the performance of multiple TCP flows in a vertical handoff. Although the performance of an individual TCP flow due to a vertical handoff has been studied in the literature, the effect of a vertical handoff on *multiple* TCP flows has been little studied. Our modelling approach adopted in Papers 1, 2 and 3 to study the TCP behaviour in vertical handoff and devise solutions to improve TCP behaviour in handoff is based on a single TCP flow. While this approach has its limitations in regard to the nature of conclusions that can be drawn, it is valuable in isolating the problems of TCP without the complexity introduced by competing flows [14]. So in this paper we show that the algorithms described in Papers 2 and 3 can be easily adapted for multiple TCP flows by scaling the TCP parameters with the number of TCP flows.

The work in this paper was carried out in collaboration with Markku Kojo. Both authors contributed equally to the design of the algorithms and the planning of the experiments. The ns-2 implementation for the simulation, the simulation experiments and the analysis of the experiments were carried out by the author with the help of the coauthor. The paper was written mainly by the author with valuable feedback and suggestions from the coauthor.

1.4 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 provides the background to the thesis in describing wireless access networks and vertical handoff. Chapter 3 describes the problems of TCP in a vertical handoff and is based on Papers 1 and 2. Chapter 4 describes our cross-layer assisted TCP algorithms for improving handoff performance and is based on Papers 2, 3 and 4. Chapter 5 gives the conclusions of the thesis and directions for future work. The four original publications which describe the research presented in this thesis are included at the end of this thesis. We have included these papers in the numbered references in this thesis and sometimes refer to them using these numbers [39, 40, 43, 44] in the context of discussion of related work.

Chapter 2

Wireless Access Networks and Vertical Handoff

This chapter gives an overview of the common wireless access network technologies along with a description of vertical handoff that can occur between any pair of access networks. Section 2.1 describes briefly five common wireless access networks, namely, Wireless LAN (WLAN), GPRS/EGPRS, UMTS, WiMAX and Mobile-Fi and a wireless overlay architecture that organizes wireless access networks in a hierarchical structure to support seamless connectivity. Section 2.2 describes vertical handoff in wireless access networks. Section 2.3 describes a simple example of a Mobile IPv6-enabled vertical handoff architecture involving WLAN and EGPRS. This example serves as a prototype for our experiments described in Chapters 3 and 4. Section 2.4 provides a summary of the chapter.

2.1 Wireless Access Networks

Wireless access networks are the end-user radio connections to the Internet. They provide the last (and/ or first) hop between a mobile node and the fixed wired Internet infrastructure using radio waves. The wireless network infrastructure consists of *base stations, access controllers, connectivity software and a distribution system.* A base station is the point of attachment that interfaces the radio signals to the wired network called the distribution system. The term base station is usually used in cellular networks whereas in wireless LANs these points of attachment are known as *access points* (APs). So the base station has a radio interface to the mobile node and also an interface to the wired network. A wireless *network interface card* (NIC) provides the interface between the mobile node and wireless

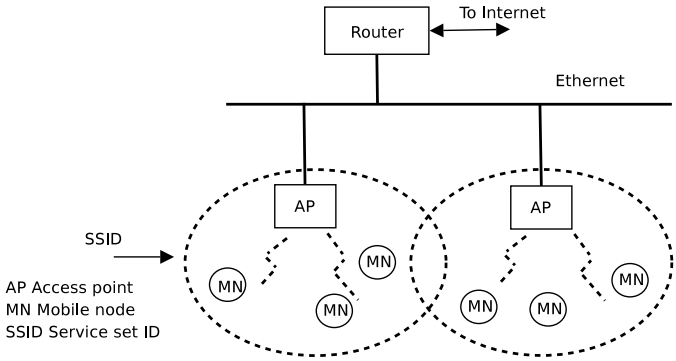


Figure 2.1: IEEE 802.11 Infrastructure mode WLAN. (Adapted from [18])

network infrastructure. A *wireless network standard* defines how a wireless NIC operates. For a mobile node to communicate with a base station both should have a wireless network interface card that implements the same link layer technology. Some terms defined here are related to wireless LANs but equivalent terms exist in cellular networks. Based on the link technologies employed, these access networks vary widely in their characteristics such as geographical range, link bandwidth, propagation delay and bit-error rate. In this section we give an overview of the common wireless access networks with an emphasis on their characteristics that influence the behaviour of TCP. A good overview of the various wireless technologies and their architectural principles can be found in [110].

Wireless Local Area Network (WLAN)

IEEE 802.11 WLAN [18, 72] has emerged as a popular wireless technology that provides broadband access within a small geographical region (a typical radius of a few tens of meters).

The basic form of IEEE 802.11 WLAN is called a *Basic Service Set* (BSS). There are two forms of BSS, *infrastructure* BSS and *independent* BSS. The infrastructure BSS shown in Figure 2.1 consists of an *Access Point* (AP) and a number of stations associated with an AP. Each AP acts as an interface between the wireline infrastructure and the wireless link with its mobile nodes (MNs). In the independent BSS, several mobile nodes can communicate simultaneously with one another in an ad-hoc mode with a direct path between any two communicating nodes.

Access points (AP) are the base stations in a wireless network. Usually

a wireline Ethernet is used to connect the multiple APs and such a system of AP networks is called a *Distribution System* (DS). APs can also be connected via wireless links. A set of BSSs along with a DS is called an *Extended Service Set* (ESS). An ESS is identified by its *service-set identification* (SSID) and an SSID is often referred to as the name of the access network. An access router is an IP router in the access network.

The IEEE 802.11 standard covers the physical and MAC-sublayer of the ISO/OSI reference model [79]. The logical link control (LLC) is defined in the IEEE 802.2 LAN standard. This allows the existing protocols to run on top of IEEE 802.11. The IEEE 802.2 LLC is also used in IEEE 802.3 (Ethernet) and IEEE 802.5 (Token Ring) LANs. So the existing protocols such as TCP/IP can be implemented over WLANs just as in wired Ethernets.

The IEEE 802.11 has many variants based on the technology used at the physical layer. The IEEE 802.11a [68] extension provides data rates ranging from 6 Mbps to 54 Mbps. The IEEE 802.11b [69] supports data rates up to 11 Mbps. IEEE 802.11g [73] is an extension to IEEE 802.11b that supports data rates up to 54 Mbps and is backward compatible with IEEE 802.11b but not with IEEE 802.11a. Another standard IEEE 802.11h [74] is an enhancement of IEEE 802.11a and it is aimed at providing data rates up to 100 Mbps. The more recent standard, IEEE 802.11n [77], envisages a maximum throughput of at least 100 Mbps. One-way propagation delay in a WLAN ranges roughly from 1 ms to 10 ms.

General Packet Radio Service(GPRS)

General Packet Radio Service (GPRS) is a packet-switched extension of *Global System for Mobile communications* (GSM) [22, 31, 122]. GSM combined with GPRS is often known as *2.5G cellular systems*. Figure 2.2 illustrates the GSM-GPRS architecture. The mobile nodes connect to the *Base Transceiver Stations* (BTS) and each BTS caters to the mobile nodes in a relatively small geographical area around it called a *cell*. Several BTSs are gathered under a *Base Station Controller* (BSC). BSC handles the access to the medium, radio resource scheduling and data transfer from/to the mobile node. Many BSCs are connected to a Mobile Switching Centre (MSC). The *Gateway MSC* (GMSC) handles the connections to a fixed network such as a *Public Switched Telephone Network* (PSTN) or *Integrated Services Digital Network* (ISDN). Several database registers are used for call control and network management.

The addition of two principal nodes, *Serving GPRS Support node* (SGSN) and *Gateway GPRS Support Node* (GGSN), enhances the GSM network to

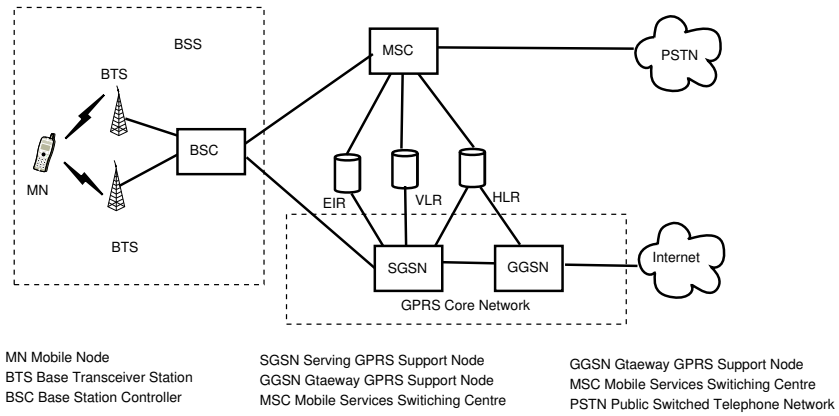


Figure 2.2: GSM/GPRS system architecture. (Adapted from [22])

provide the upgraded packet switching service to GPRS. SGSN handles the routing of packet-switched data to and from the mobile node, user mobility, logical link management and the authentication and the accounting functions. The GGSN provides the connectivity to an external packet network such as Internet and X.25. The GPRS protocol stack follows a layered architecture up to layer 3 of the ISO/OSI reference model.

GPRS can provide a data rate up to 171 Kbps (theoretical maximum) [22] with a one-way propagation delay of about 300 ms. The *Enhanced GPRS* is called EGPRS or *Enhanced Data Rate for GSM Evolution* (EDGE) [139] provides a transmission rate ranging from 128 Kbps to 473 Kbps (theoretical maximum) [104].

Universal Mobile Telecommunication System (UMTS)

UMTS, a third-generation (3G) cellular network [5–9] consists of three interactive domains: *Core Network* (CN), *UMTS Terrestrial Radio Access Network* (UTRAN) and *User Equipment* (UE). The UMTS Core Network is based on the GSM Phase 2+ network with GPRS. The UTRAN air interface is based on *Wideband CDMA* (WCDMA) technology and it can provide a data rate that ranges from 384 Kbps to 2 Mbps with a link delay around 150 ms [61].

UMTS *High-Speed Packet Access* (UMTS/HSPA) [3] is a combination of two technologies, High-Speed Downlink Packet Access (HSDPA) [11] for downlink and High-Speed Uplink Packet Access (HSUPA) [1] for the uplink.

HSPA can provide a data rate up to 14 Mbps with a one-way propagation delay that ranges from 25 ms to 50 ms. HSPA+ [2], also known as *evolved HSPA*, is an enhanced version of HSPA that can provide data rates up to 42 Mbps. UMTS *Long Term Evolution* (LTE) [4, 38, 123] is a part of GSM evolution beyond 3G and the objective of LTE is to provide an extremely high performance radio-access technology that offers full vehicular mobility and can coexist with all GSM/UMTS technologies. LTE can provide data rates up to 100 Mbps with a one-way propagation delay less than 10 ms.

Worldwide Interoperability for Microwave Access (WiMAX)

IEEE 802.16 [48, 70, 75] is an emerging standard for global broadband wireless access and it supports fixed, nomadic, portable and mobile operations. *Worldwide Interoperability for Microwave Access* (WiMAX) Forum [152] is an organization that promotes conformance and interoperability of the IEEE 802.16 standard. WiMAX supports data rates from 1 Mbps to 40 Mbps and has a one-way propagation delay of less than 100 ms. *Mobile WiMAX* [153, 154] is compatible with the IEEE 802.16e standard.

WiMAX can operate in *fixed access*, *nomadic*, *portable* and *mobile* modes. In the *fixed access network*, the access is provided through a fixed antenna and it does not support portability and handoff between APs. The *nomadic mode* allows a terminal to access the operator's network from different APs. The *portable mode* of operation allows handoff at walking speeds. In all the three modes WiMAX can provide a data rate of 40 Mbps within a cell radius of 10 kilometers. The fully mobile mode allows seamless connectivity in a mobile environment and its aim is to provide low latency, low packet loss and real-time handoffs between APs at vehicular speeds of 120 km/hour or higher. It can support a data rate of 15 Mbps with a one-way propagation delay less than 100 ms.

Mobile Broadband Wireless Access (MBWA) - Mobile-Fi

Mobile Broadband Wireless Access (MBWA), also known as Mobile-Fi, is IEEE 802.20 in its technical specifications and is optimized for the data transport of IP-based services [28, 71]. It promises a ubiquitous, always-on and inter-operable multi-vendor mobile broadband wireless access network. Mobile-Fi can deliver broadband Internet access with peak data rates greater than 1 Mbps with one-way propagation delay less than 10 ms. It can support vehicular mobility classes up to 250 km/hour.

Table 2.1 provides a summary of the key characteristics of the various wireless access networks that we have described here.

Table 2.1: Comparison of the Wireless Access Network Technologies

Access Network	Standard	Data rate	One-way propagation delay	Mobility
WLAN	IEEE 802.11b	up to 11 Mbps	1-10 ms	Low
	IEEE 802.11a	up to 54 Mbps	1-10 ms	Low
	IEEE 802.11g	up to 54 Mbps	1-10 ms	Low
	IEEE 802.11h	up to 100 Mbps	1-10 ms	Low
	IEEE 802.11n	up to 540 Mbps	1-10 ms	Low
2G	GSM	up to 57.6 Kbps	350 ms	High
	GPRS	up to 171 Kbps	300 ms	High
	EDGE/EGPRS	up to 473 Kbps	300 ms	High
3G	UMTS	384 Kbps	150 ms	High
	HSPA	up to 14 Mbps	25-50 ms	High
	HSPA+	up to 42 Mbps	25-50 ms	High
	LTE	up to 100 Mbps	$<$ 10 ms	High
WiMax	IEEE 802.16	up to 40 Mbps	50 ms	High
Mobile-Fi	IEEE 802.20	$>$1 Mbps (peak)	$<$ 10ms	High

Wireless Overlay Networks

As can be seen in Table 2.1, no single wireless access network is capable of providing a high bandwidth, low propagation delay and wide-area connectivity to a large number of mobile users simultaneously. For example, the cellular networks such as GPRS and UMTS provide a wide-area connectivity with low data rates to users with high mobility. On the other hand WLAN offers much higher data rates to users with low mobility.

A mobile node equipped with radio interfaces to WLAN and GPRS/UMTS allows a user to combine the benefits of the high data rate of WLAN whenever it is available with the always-on connectivity of GPRS/UMTS. An architecture which provides an integration of a variety of wireless access networks with diverse characteristics by combining them in a hierarchical order ranging from a low-delay, high-bandwidth and short range access network to a high-delay, low-bandwidth and wide range access network is called a *wireless overlay network* [145]. A vertical handoff allows a mobile user to roam across an overlay network by seamlessly connecting to the appropriate wireless access network with little disruption in connectivity and in a manner transparent to the application.

2.2 Vertical Handoff

When a mobile node moves away from a base station (or access point), the signal strength deteriorates and there is a need to switch to another base station from where it gets an adequate stronger signal. Handoff (handover) is the mechanism by which an ongoing connection between a mobile node and a *correspondent node*, the entity in communication with the mobile node, is transferred from one base station to another [100, 117]. Handoff can be classified as *horizontal handoff* (intrasystem) and *vertical handoff* (intersystem) [100, 111, 145].

A horizontal handoff involves a mobile node moving between access networks that use the same link layer technology, for example, from UMTS to UMTS or from WLAN to WLAN. The need for a horizontal handoff arises, for example, when the signal strength of the serving base station deteriorates below a certain threshold value.

A vertical handoff arises when a mobile node moves out of the serving network to another access network of a different link technology, for example, a handoff between WLAN and UMTS. A wireless overlay network is the conceptual architecture for vertical handoff [145]. A vertical handoff, as in any handoff scenario, can be of two types based on the connectivity to the old access router during the handoff, namely, *break-before-make* (BBM) and *make-before-break* (MBB) [100]. In a break-before-make handoff, the mobile node's connection to the old access router ends before the completion of the handoff, thereby causing disruption in connectivity which often results in packet loss. In the worst case, an entire window of TCP segments may be lost. By contrast, in a make-before-break handoff, a mobile node can have connection to more than one access router at the same time and the mobile node ends its connection to the old access router only after establishing

the connection to the new access router, thereby avoiding packet losses due to a handoff. In a make-before-break handoff, the mobile node can use both the old and the new interfaces to send and receive packets simultaneously.

Possible reasons for the occurrence of a break-before-make handoff include (i) a network interface card that is able to connect to different radio networks but only to a single network at a given time, e.g., a mobile phone hardware that can switch between GPRS and UMTS (ii) there is no overlapping coverage area, e.g., the mobile device is connected to an access network A and it passes through a tunnel where there is a disruption in connectivity and at the end of the tunnel the mobile device is connected to another access network B which uses a different link layer technology (iii) even if there is an overlapping coverage, the user may not have access rights to wireless network, e.g., a mobile device moving from access network A to access network C and it has access rights to A and C and have no access rights to the network B which comes in between A and C. In addition to the above scenarios, a break-before-make handoff can occur in the case of a mobile device with two or more radio interface cards if there may not be enough power from the battery to have two radios on simultaneously.

2.3 A Vertical Handoff Architecture with WLAN and EGPRS

In this section we describe a vertical handoff architecture by taking a concrete example of a vertical handoff between WLAN and EGPRS using Mobile IPv6 as the mobility protocol to illustrate the concepts involved in this architecture. There are two main internetworking approaches for integrating WLAN and EGPRS/UMTS [10, 32, 127, 140, 157], namely, *tightly-coupled architecture* and *loosely-coupled architecture*. In a tightly coupled architecture, a WLAN appears as another 3G access network to the 3G core network providing a seamless service continuation across 3G and WLAN. As the WLAN is connected to a 3G core network, it requires that the same access provider must own the WLAN and the 3G parts of the network. In the loosely-coupled architecture, the WLAN is connected to the Internet through the WLAN gateway and it does not have any direct connection to the 3G core network or to any 3G network element. In this architecture, the WLAN and the EGPRS/UMTS can have different access providers and the IP addresses of these two access networks will be different. So when a mobile node switches between these access networks some kind of mobility mechanism is needed as the two access networks have different IP prefixes.

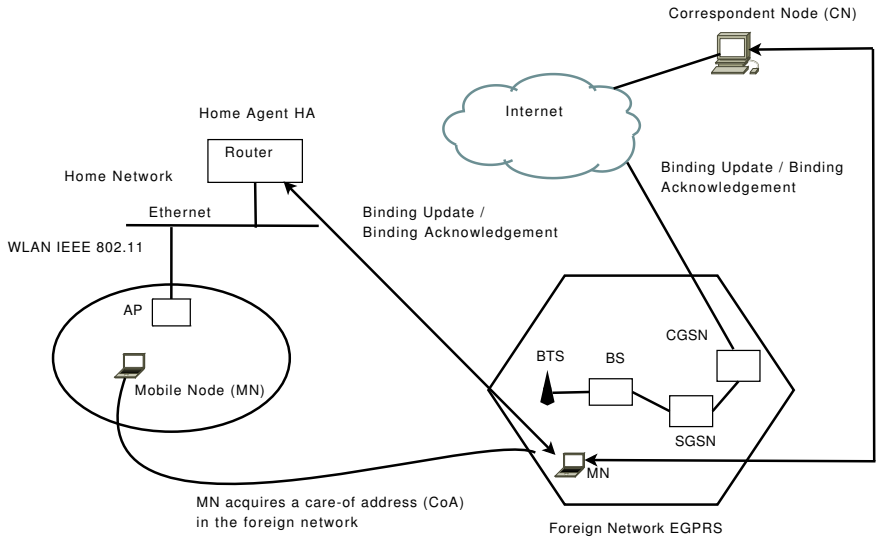


Figure 2.3: Vertical handoff Architecture from WLAN IEEE 802.11 to EGPRS using Mobile IPv6. Adapted from [157]

In the Internet, any device is identified by its IP address. Also at the network layer, the IP address is used for routing i.e., the IP address is associated with a fixed network location. This dual role played by the IP address necessitates the reconfiguring of the IP address when a device moves from its original location (home network) to another location (foreign network) in order to support seamless communication [23]. As TCP uses the IP address and the port numbers to identify a session, the reconfiguration of the IP address as the device changes its location should be transparent to TCP and to the higher layers.

In order to achieve mobility between different IP networks we need protocols that support mobility which are implemented above the link layer. While Mobile IPv4, Mobile IPv6 [84, 116] (implemented at the IP layer) and TCP-Migrate [142] (implemented at the transport layer) are some of the well-known mobility protocols, protocols such as Host Identity Protocol (HIP) [105, 107, 120] (implemented at the Host identity sublayer - layer 3.5), SCTP [147] and Multipath TCP [54, 62] (implemented at the transport layer) and Session Initiation Protocol (SIP) [63, 136] (implemented at the application layer) are capable of supporting mobility. Proxy Mobile IPv6 [57] is a network based mobility protocol that extends Mobile IPv6 functionalities in which a proxy mobility agent in the network does the mobility signalling with the home agent on behalf of the mobile node

attached to the network. Mobility-related protocols which are implemented above the IP layer rely on other layers for location management, e.g., DNS lookup. A discussion of mobility protocols based on the layer of their operation in the protocol stack is described in [21]. The considerations for the choice of the layer in implementing a mobility protocol along with the design tradeoffs involved in the choice are discussed in [46].

Figure 2.3 shows a loosely-coupled vertical handoff architecture integrating WLAN and EGPRS using the Mobile IPv6 as the mobility protocol. We model this architecture in the simulation experiments described in this thesis. Instead of Mobile IPv6, it is also possible to use protocols such as HIP, SCTP and SIP which have support for mobility. In a vertical handoff involving access networks such as EGPRS, UMTS and UMTS/HSPA, the mobility may be limited to subnetwork mobility (below IP layer) and it is transparent to the network layer and the upper layers as the mobile node changes its point of attachment solely using the link layer mechanisms. The study of the different internetworking architectures and the various mobility and handoff signalling schemes is not within the scope of the work described in this thesis.

Mobile IPv6 is an IP layer mobility protocol based on IPv6. In Mobile IPv6, a mobile node is identified by a permanent address called the *home address* and is associated with a *home network*, a network having a network prefix matching that of the mobile node's home address. The *home agent* (HA) is the router that supports mobility in the home network. When a mobile node moves to another network called the *visited network*, the *foreign agent* (FA) is the router that provides routing services to the mobile node in the visited network. When a mobile node is in the home network, conventional methods are used to route packets addressed to it. In the visited network, the mobile node obtains a temporary address called the *care-of address* (CoA) and informs the HA about it. The mobile node detects that it has moved to a visited network by analyzing the *Router Advertisement* it receives from the access routers (AR) and forms a care-of address.

The association between the home address and the care-of address is called *binding*. The mobile node does a binding registration by sending a *binding update* message that supplies the care-of address to both the home agent and the correspondent node. In this example, we assume that Mobile IPv6 with *route optimization* where a mobile node can directly communicate with a correspondent node is used. The home agent and the correspondent node respond to the mobile node with a *binding acknowledgement* message. At this point the mobile node can communicate with

the correspondent node using the new (visited) wireless access network.

A handoff process involves two steps: a *handoff decision phase* and a *handoff execution phase*. At first, either the mobile node or the network or perhaps both together have to decide when to perform a handoff. This process is called the handoff decision process. The handoff execution process comes after the decision process. The various considerations involved in these mechanisms are described in [111].

The delay arising from a handoff is due to three main components: *detection period*, *address configuration interval* and *network registration time* [36]. The detection time is the time taken by the mobile node from discovering that it is under the coverage of a new access point by using, for example, link-layer beacons or Probe Requests to the time it receives the Router Advertisement from the new access router. Address configuration time is the interval from the time a mobile node receives the Router Advertisement to the time it takes to update its routing table and assign a new care-of address. The network registration time is the time taken by a mobile node from sending a binding update to the home agent as well as to the correspondent node until the receipt of the first packet from the correspondent node. This is so because Mobile IPv6 does not specify (as it is optional) that the mobile node should wait for the binding acknowledgement from the correspondent node. As the connection to the new router will be operational after a handoff delay, this period can be taken as the disconnection period for a break-before-make handoff.

Due to the signalling overhead in Mobile IP and the link switching involved, the resulting handoff delay may be quite large. Fast handoff techniques to reduce the handoff delay are described in [49, 91]. A detailed description of mobility in the Internet, Mobile IP protocols, inter-networking with Mobile IP and fast handoffs can be found in [119].

2.4 Summary

This chapter describes the background material on wireless access networks and vertical handoff that provides the context for our work. Wireless access networks are the end user radio connections to support mobility in the Internet. We briefly described WLAN, GPRS, UMTS, WiMAX and MBWA which are among the common wireless access technologies that have evolved in the recent past. With the proliferation of wireless access to the Internet, wireless access technologies play a basic role in the Internet infrastructure. A comparative view of the characteristics of the different wireless access networks given in Table 2.1 shows a wide disparity in their

data rate, propagation delay and mobility. We briefly described the wireless overlay architecture, a conceptual scenario for vertical handoff. We introduced vertical handoff and illustrated a handoff architecture involving WLAN and EGPRS with Mobile IPv6 as the mobility protocol. This handoff model is used in our simulation experiments described in Chapters 3 and 4.

Chapter 3

Analysing TCP Behaviour with Vertical Handoff

In this chapter we describe the behaviour of TCP in a vertical handoff. We begin this chapter with a brief description of TCP congestion control. Section 3.2 describes the organization of our simulation experiments and the simulation model. Section 3.3 describes the problems of TCP due to a vertical handoff and it is based on Papers 1, 2 and 3. Section 3.4 describes some example scenarios that show the effect of a vertical handoff on TCP performance. Section 3.5 provides a summary of the chapter.

3.1 TCP Congestion Control Algorithms

TCP [118], is the most widely used transport protocol in the Internet, is used by applications such as Web browsing, e-mail, file transfer and streaming audio and video and it accounts for a bulk of the Internet traffic [34, 35, 37, 128, 144, 148, 158]. TCP is an *end-to-end protocol* as it is implemented at the end hosts and the routers inside the network do not have any knowledge (state) of the protocol. TCP performs the twin goals of reliable data transfer and congestion control [16, 81, 118].

The congestion control algorithm used in TCP was essentially introduced by Van Jacobson in a seminal paper in 1988 [81]. The purpose of the TCP congestion control algorithm is to share the available bandwidth in the network among the various flows in a fair and efficient manner. Efficient utilization of the network can be achieved by utilizing the links close to their capacities and keeping the router buffers small. Fairness is more complex to define precisely. Intuitively fairness can be interpreted to mean that no flow is unduly starved. Studies have shown that the fairness achieved by

TCP comes close to *proportional fairness* which accounts for TCP's bias against large RTT flows, i.e., flow rates tend to be allocated such that flows with larger RTT have smaller rates [143].

TCP belongs to a generic family of sliding window protocols that provides a *reliable, in-order, full-duplex, byte-stream* delivery of data to the applications [118]. The basic unit of data transfer, known as a *segment*, is a contiguous sequence of bytes. Each byte is identified by a 32-bit *sequence number*. Each segment can be of size less than or equal to the *maximum segment size* (MSS) which is negotiated by the TCP sender and the TCP receiver at the beginning of a connection. The TCP receiver sends a *cumulative acknowledgement* for byte k to the TCP sender which indicates that all the bytes with lower segment number than k have been received successfully and that it is expecting a segment starting from byte k. The *delayed acknowledgement* mechanism [30] allows a TCP receiver to refrain from sending an acknowledgement (ACK) for every segment received, usually acknowledging every second segment, which results in sending fewer ACKs thereby saving bandwidth and processing. When the TCP receiver receives an out-of-order segment it will send a duplicate ACK (*dupack*) for the last received in-order segment.

TCP congestion control algorithm uses a *window* known as the congestion window (*cwnd*) to determine the maximum number of unacknowledged (outstanding) packets a connection can have in the network. By dynamically varying the window based on the information from the network about the level of congestion in the network, TCP seeks to adapt its sending rate to its proper share of the available network bandwidth. The receiver advertised window (*rwnd*) which reflects the amount of buffer space available at the TCP receiver sets an upper bound on the size of the *cwnd*.

At the beginning of a TCP connection, TCP starts to probe the unknown network to determine the available capacity so that it should not congest the network by sending a large burst of data. The *slow-start* phase governs the evolution of the TCP window at the beginning of a connection. The window starts at a small value, either one or a few segments, and increases by one segment for each acknowledgement (ACK) received, thereby doubling the size of the window size for every RTT. This exponential growth of the window during the slow start phase continues until either the window size reaches a value determined by the variable called *slow start threshold* (*ssthresh*) or a packet loss occurs. The optimal value of the initial *ssthresh* is the available capacity in the end-to-end path of the TCP connection which is the product of the bottleneck link bandwidth times the RTT of the end-to-end path [65]. The subsequent evolution of

the window is governed by the *congestion avoidance* phase. The initial *ssthresh* is set to a high value not exceeding the *rwnd* and is reduced when a packet loss occurs. In the congestion avoidance phase the window grows more cautiously and for each acknowledgement received, it increases by the reciprocal of the current window size. So the window roughly increases by one segment in a round-trip time.

TCP considers a packet is lost when it receives a *dupthresh* number of duplicate acknowledgements (*dupacks*) or when a retransmission timeout occurs. As *dupacks* can arrive due to packet reordering, TCP waits till it gets a certain number of *dupacks*, represented by *dupthresh*, to infer a packet loss. RFC 5681 [16] defines *dupthresh* to be 3. For each segment sent, the TCP sender waits for the acknowledgement for a certain duration called *Retransmission Timeout* (RTO) and retransmits the segment if the acknowledgement has not arrived before the expiry of the RTO timer. TCP interprets a packet loss detected by an RTO as an indication of severe congestion in the network. By way of its response, TCP retransmits the lost packet, reduces the sending rate by setting the *cwnd* to one segment and the *ssthresh* to half number of outstanding packets (*FlightSize*), and enters slow start.

When the loss is detected via *dupacks*, TCP retransmits the missing packet and sets *cwnd* to one segment and *ssthresh* to half the *FlightSize*. This is the *fast retransmit* algorithm implemented in TCP Tahoe [81]. After the fast retransmit, TCP Tahoe enters slow start. As a result the *pipe*, which refers to the TCP sender's estimate of the number of outstanding segments, is emptied fully and all the outstanding segments are retransmitted. This is especially inefficient in high bandwidth-delay product networks.

TCP Reno [82] is similar to TCP Tahoe except for the difference in behaviour after a fast retransmit. After a fast retransmit, TCP Reno sets the *cwnd* and *ssthresh* to half the *FlightSize* and enters fast recovery. At the beginning of fast recovery, TCP Reno inflates the *cwnd* by three segments (i.e., *ssthresh* +3) assuming that three segments have already left the network as TCP has received three *dupacks*. For each additional *dupack* that arrives, *cwnd* is incremented by one segment and a new segment is sent if the new *cwnd* and the receiver window allow it. This implies that the sender effectively waits for half a window of *dupacks* before it can send a new segment. Upon reception of the first acknowledgement that acknowledges the new data, the *cwnd* is deflated back to *ssthresh*. This ends the fast recovery and TCP continues in congestion avoidance phase. The advantage of TCP Reno is that instead of flushing the pipe completely as in TCP Tahoe, it reduces the pipe by half. TCP Reno performs well

when a single packet is lost in a window but RTO recovery may be needed for recovering multiple packet losses in a window.

TCP NewReno [52, 65] enhances the fast recovery algorithm of TCP Reno. At the time the recovery starts, it keeps a variable *recover* to denote the highest sequence number transmitted. NewReno introduces the concept of *partial ACK* which acknowledges a retransmitted segment but not all the segments up to *recover*. The behaviour of NewReno is similar to Reno except for its behaviour in regard to partial ACKs. In fast recovery, upon the receipt of a partial ACK, NewReno retransmits the first unacknowledged segment and sends new data if window allows it, as in the case of Reno. NewReno can recover from multiple packet losses if there are multiple packet losses in a window but performance degrades especially in the case of long-delay networks as NewReno can retransmit only one lost packet per RTT.

The use of the Selective Acknowledgements (SACK) option [101] significantly improves TCP's loss recovery mechanism especially when there are multiple losses in a single window. In TCP SACK, the TCP receiver, along with an acknowledgement can send information up to four non-contiguous segments received beyond the first missing segment. The TCP sender can retransmit the missing segments based on this information. As TCP SACK algorithm [27] incorporates all the key error-recovery mechanisms, it is efficient [50] and it is the most widely deployed TCP in the Internet [103, 109].

As packet loss is interpreted as the congestion signal, the TCP algorithm has no way of distinguishing the packet losses due to wireless channel errors from packet losses due to congestion. The choice of packet loss as the congestion indicator might seem appropriate in an era when most of the links in the Internet were wired links. Besides, the routers have to do very little else than to drop the packet when the router buffers overflow. In wireless and mobile environments this assumption no longer holds due to the high bit error rate, link outages, disconnections and handoffs. As a result, the congestion control response of TCP in wireless networks can be unnecessary and inefficient. RFC 3481 [78] describes the problems that TCP encounters in 2.5G and 3G wireless networks and recommends enhancements to adapt TCP to the specific characteristics of the wireless and mobile networks.

3.2 Organization of the Simulation Experiments

In order to understand the effects of a vertical handoff on TCP behaviour we conduct experiments using the ns-2 simulator [106]. The vertical handoff architecture is assumed to be a loosely-coupled architecture integrating two

wireless access networks with Mobile IPv6 as the mobility protocol and it is similar to the integrated WLAN-GPRS architecture shown in Figure 2.3 of Chapter 2. Using the same architecture, we evaluate the performance of TCP with the enhancements we propose in this thesis. Since we are interested in evaluating the behaviour of TCP in a vertical handoff, the two wireless access links involved in the handoff are represented by their bandwidth and the propagation delay, the two characteristics that affect the TCP behaviour. We often refer to the propagation delay of an access link as the 'delay' of the link. We use the notation ' x/y link ' to denote a link of bandwidth x and delay y. For instance, a link designated as 100 Mbps/2 ms denotes a link whose bandwidth is 100 Mbps and propagation delay is 2 ms. We also use this notation to refer to a link whose attributes are specified non-numerically, for instance, a high-bandwidth/low-delay link. Mobile IPv6 is not simulated but we model that a handoff notification with the characteristics of the access link will arrive at the TCP sender at the correspondent node along with the binding updates. Handoff notification with the binding updates is modelled using a user-defined command in ns-2 with the link characteristics as its parameters.

Simulation Model

The basic idea behind a simulation model for vertical handoff is to provide multiple paths with different characteristics between a mobile node and its correspondent node.

In the simulation models used in [58,59], a vertical handoff is simulated by changing the link characteristics such as link bandwidth, propagation delay and buffer size so that the same link models the characteristics of both the links involved in the handoff. When the buffer size is changed, the default ns-2 behaviour is not to discard packets which cannot be accommodated within the size of the new buffer. This is the approach used in the aforementioned papers to simulate a make-before-break handoff where there are no loss of packets during the handoff. To model a break-before-make handoff where packets may be lost during the handoff, in ns-2 it is possible to apply an error model with 100 % (or arbitrarily specified loss percentage) loss on the link which is the approach taken in [58,59]. In our view, there are many drawbacks in this approach. In a make-before-break handoff, a mobile node can continue to receive packets for a while from both the old link and the new link. So modelling the same link to represent both the links makes it difficult to distinguish the packets arriving through these links and treat them separately if needed. Another problem with this approach is that after a handoff, a link will always have the packets of the

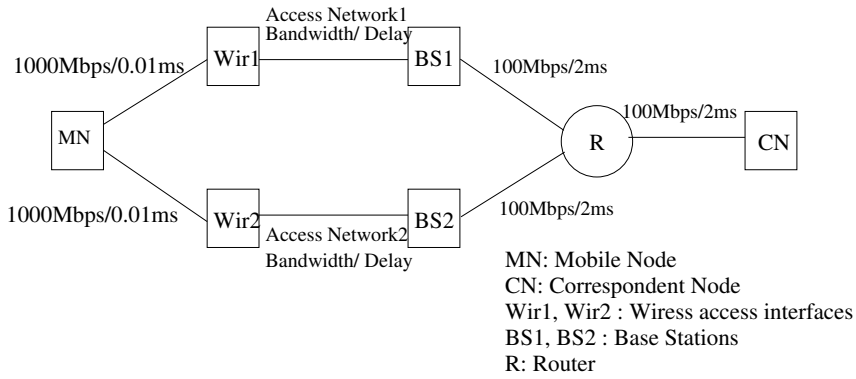


Figure 3.1: Network Topology used in Vertical Handoff Experiments.

old link and it may be difficult to simulate a new link with an empty buffer. Yet another problem is that it is not possible to represent the background traffic of the old link and the new link independent of each other.

The simulation model we propose here and use in our experiments in this thesis reflects a vertical handoff more realistically by changing the routes the packets take before and after a handoff rather than by merely changing the link characteristics. We use the ns-2 routing features to this end as ns-2 supports routing updates on-the-fly. By changing the route metrics of the two links involved in a handoff, packets can be directed along different links. The route metric of the current route is set to a very low value while that of the unused route is set to a very high value. When a handoff notification arrives, the route metric is changed to model a make-before-break handoff. To model a break-before-make handoff, an error model with a packet loss rate of 100 % is applied to the old link and after the disconnection period, the route metric of the new link is set to a small value.

The simulation model used in our experiments is shown in Figure 3.1. The mobile node MN is capable of switching between the two wireless access interfaces, namely, Wir1 and Wir2. Both Wir1 and Wir2 have dedicated base stations BS1 and BS2 that are connected to a common access router R which has a link to the correspondent node CN. The nodes Wir1 and Wir2 are introduced between the MN and the respective base stations to correctly model a break-before-make handoff. Without Wir1 (Wir2), if we apply an error model with a loss rate of 100 % to BS1 (BS2) node, only the packets arriving at BS1 (BS2) are dropped and the packets in transit from BS1 (BS2) to MN will not be dropped. When we apply this error model to both BS1 and Wir1 simultaneously, the link between BS1 and Wir1 breaks and the packets that have arrived at BS1 and those in transit between BS1

and Wir1 are lost. As a result MN loses the connection to the base station BS1 thereby modelling the break-before-make handoff accurately. The link between MN and Wir1 (Wir2) has "infinite buffer" and "no" propagation delay, i.e., Wir1 and Wir2 are local to MN and do not contribute to the delay of MN's path to the base station. The propagation delay and the bandwidth of the fixed links are as shown in Figure 3.1.

Simulation Setup

In order to study the effect of changes in the link bandwidth and delay on TCP behaviour we categorize our experiments into the following three classes: (1) handoff between access links which have the same bandwidth but different delay, (2) handoff between access links which have the same delay but different bandwidth, and (3) handoff between access links which have the same bandwidth-delay product (BDP) but with differing bandwidth and delay. The choice of the parameters for the bandwidth and delay cover the entire range of values that are of interest in typical handoff scenarios.

In our experiments, we use TCP SACK as the baseline TCP to evaluate its performance in a vertical handoff and to quantify the improvements in TCP performance due to the use of our algorithms. We refer to TCP SACK as regular TCP or just 'TCP' in presenting our experimental results. The TCP initial window of three 1460-byte segments is selected based on RFC 3390 [15]. Delayed ACK [30], Limited transmit [13] and DSACK [53] are enabled in TCP. The receiver advertised window is set to 5000 packets so that it will not be a limiting factor in setting the congestion window. The TCP packet size is 1500 bytes inclusive of the TCP/IP headers. For the experiments reported in Paper 1, the WLAN buffer is set to 30 packets and the EGPRS buffer is set to 32 packets to closely characterize the WLAN, EGPRS parameters. Channel allocation delays in EGPRS are included in this paper. For the experiments in Papers 2, 3 and 4, the router buffer size of each link is set to the BDP of the access link with at least five packets as the minimum value and no allocation delay is modelled.

In our experiments, only one mobile node and one correspondent node are considered. One to four bulk TCP flows are transferred from the correspondent node to the mobile node. The small number of multiple flows in the study is adequate considering that a mobile device typically runs a small number of applications simultaneously. In our experiments a handoff can occur once in the lifetime of a TCP connection in any of the TCP phases, namely, slow start, slow start overshoot, fast retransmit/fast recovery and congestion avoidance. In our experiments, a 20-second interval is

chosen to cover all the phases of a TCP connection and a handoff can occur uniformly in any of the 200 points at 100 ms intervals once in the lifetime of a TCP connection. Each experiment is conducted 200 times corresponding to the handoff occurring at these 200 points in the 20-second interval. The duration of each test run includes the completion of the handoff occurring in the 20-second interval.

Both make-before-break as well as break-before-make handoffs are examined. The disconnection period for break-before-make handoff is taken to be 500 ms. As discussed in Section 2.3, the handoff delay which is the disconnection period for a break-before-make handoff consists of the time taken for network discovery, authentication and for sending the binding updates. In our experiments we take the handoff delay to be the minimum time for the binding updates from the mobile node to reach the correspondent node and it corresponds to the one-way propagation delay of the new path after the handoff. We choose 500 ms as the disconnection period which is greater than the largest one-way propagation delay (300 ms) of any of the access links used in our experiments.

No link errors are modelled in our simulations as we assume that the packet losses are either due to disconnection or congestion. This choice is made as the objective of the present study is to isolate the effects of a vertical handoff on TCP.

Performance Metrics

As we are interested in studying the behaviour of TCP with a vertical handoff, we study how TCP behaves immediately after the occurrence of a handoff. As a performance index, we calculate the time taken to transfer (until acknowledgement is received) 'n' new data packets through the new access link after a handoff. The choice of n depends on the severity of the effect of a vertical handoff and the value of n is chosen to reflect the time that TCP can take to stabilize after a handoff. We have carried out TCP performance analysis as n takes values from 50 to 500 in steps of either 50 or 100 packets. Experimental observations show that in many handoff scenarios, the choice of the value of n as 100 is adequate and for a handoff to a high-bandwidth/low-delay access link, e.g., handoff to a 54000 Kbps/4 ms link, choosing n to be 500 gives a better evaluation of the proposed enhancements.

The performance metric in [64,124] is the number of in-order data packets received in four seconds after a vertical handoff, chosen on the basis that for a target link bandwidth greater than 256 Kbps, the TCP sender would have recovered fully in 4 seconds. In our experimental setup some of the

links' bandwidth is less than 256 Kbps (e.g., 200 Kbps) and we have observed that four seconds is not enough for TCP to recover fully in many handoff scenarios. When the target link bandwidth is very high such as in a 54000 Kbps/4 ms access link, TCP recovery will be complete much before four seconds after a handoff. So we have chosen to consider the time taken for the transfer of n new data packets immediately after a handoff as a better metric than the number of packets transferred in a fixed duration to evaluate the effectiveness of the TCP recovery in a handoff. In all our experiments and analysis described in this thesis, we use the term *transfer time* of a packet to denote the time interval from the time the packet is sent from a TCP sender to the time the sender receives the acknowledgement for that packet from the TCP receiver.

3.3 The Problems of TCP in a Vertical Handoff

We recall that a vertical handoff involves an active mobile node switching to a new access network whose link characteristics differ significantly from that of the old access network. For example, if we consider a WLAN of bandwidth 5 Mbps and one-way propagation delay of 10 ms and an EGPRS with 200 Kbps and 300 ms as the corresponding quantities, the ratio of change in bandwidth after a vertical handoff from WLAN to EGPRS is 25:1 whereas the ratio of change in delay is 1:30. So there is an order of magnitude change in bandwidth and delay after a vertical handoff from WLAN to EGPRS.

Though the mobility management protocols such as Mobile IP [84, 116] provide a transparent mobility to the transport layer, the order of magnitude change in bandwidth and delay in the last-hop or first-hop link characteristics due to a vertical handoff may change the end-to-end path characteristics significantly and it may take TCP several RTTs to adapt to the characteristics of the new end-to-end path. Since immediately after a vertical handoff, TCP still relies on the *cwnd* and RTT of the old path, it may unnecessarily invoke congestion control actions resulting in performance degradation.

TCP is known to converge slowly to the new network conditions after a vertical handoff [47]. TCP can only increase the sending rate by one segment per RTT in the congestion avoidance phase. If the new link after a handoff has a high capacity in comparison to that of the old link, it will take TCP many RTTs to achieve the corresponding sending rate. Similarly after a handoff to a low capacity path, TCP halves the *cwnd* when packet losses occur. The new path capacity may still be lower or higher than half

of the old path capacity resulting in under (or over) utilization of the new path. A significant change in RTT can give rise to spurious retransmission timeouts (RTOs), delayed timeout recovery and packet reordering. As a result, a vertical handoff may incur packet losses, intermittent connectivity, packet reordering and spurious RTOs resulting in either unnecessary TCP congestion response or inefficient loss recovery that affects TCP performance [40, 43, 59, 64, 67, 87, 88, 133, 137, 138].

We next describe the various problems of TCP in different vertical handoff scenarios. While many of these problems have been known in the literature previously, our systematic exploration in Paper 2 has led to identifying the specific scenarios in which they occur.

Spurious Retransmission Timeouts (RTOs)

Spurious RTOs are the unnecessary retransmission timeouts caused by delayed or lost acknowledgement. It has been observed that even when no segments are lost, a TCP retransmission timer can expire spuriously and cause unnecessary retransmissions [60, 97, 98]. As a result of a spurious RTO, the TCP sender retransmits packets unnecessarily and decreases the sending rate by reducing the *cwnd* to one MSS and *ssthresh* to *FlightSize*/2. Bandwidth is wasted due to unnecessary retransmissions whereas unnecessary *cwnd* reduction results in severe underutilization of the link.

In a vertical handoff, spurious RTOs occur when a make-before-break handoff takes place from a low-delay link to a high-delay link. After the handoff, the ACKs will be delayed due to the high propagation delay of the new high-delay link. Due to the small RTO value calculated on the basis of the old path, the TCP retransmission timer expires before the arrival of the ACKs through the new link. Usually in a make-before-break handoff no packets are lost and after a spurious RTO, the late ACKs of the original packets trigger unnecessary retransmissions of a whole window of segments during the RTO recovery. After a spurious RTO, the TCP sender in slow start increases the *cwnd* on each late ACK received, thereby injecting a large number of segments in one RTT violating the packet conservation principle [81]. This situation can lead to severe packet losses if the handoff is from a high-bandwidth link to a low-bandwidth link even if the BDP of the two links remains the same. An example scenario is shown in Figure 8a, Paper 2.

TCP Eifel [97, 98] and Forward RTO-Recovery (F-RTO) [131, 132] are two well-known algorithms to detect spurious RTOs. The occurrence of spurious RTOs due to a vertical handoff has been the subject of several studies including our own [40, 43, 64, 67, 95, 125]. Our experiments described

in Paper 2 show how sensitive the occurrence of spurious RTOs is to the access link characteristics. We observe that the spurious RTOs occur in more than 85-90% of the handoff points when the delay of the new link is at least eight times that of the old link. From our experiments we find that the occurrence of spurious RTOs is dependent on the TCP state as well. For instance, if TCP enters fast recovery just after a handoff, spurious RTOs can occur due to queueing delay in the new link even with a two-fold increase in the delay of the new access link. An example scenario is shown in Figure 10a, Paper 2.

Packet Reordering

Packet reordering occurs when a make-before-break handoff takes place from a high-delay link to a low-delay link. During a make-before-break handoff, a mobile node can receive packets through the old link as well as through the new link. The packets with high sequence numbers sent after the handoff through the new link arrive at the TCP receiver earlier than the packets sent before the handoff through the old link. Reordering occurs at the receiver as the sequence number of the packets arriving through the new link is greater than the expected sequence number in the TCP variable *rcv.nxt*. As a consequence of this reordering, the TCP receiver sends duplicate acknowledgements (*dupacks*) over the new link. When the TCP sender gets three *dupacks*, it triggers the *fast retransmit* and *fast recovery* algorithms and as a result the *cwnd* and the *ssthresh* are reduced. As the *dupacks* arise, not due to congestion but due to reordering, the retransmissions are unnecessary. The *cwnd* reduction is undesirable if the BDP of the new path is greater than that of the old path. Packet burst can occur after a reordering event [64]. When the last in-order packet arrives at the receiver, it advances the TCP variable *rcv.nxt* by the amount of packets received through the new link and the receiver sends a cumulative ACK through the new link. Upon receiving this ACK, the TCP sender sends a burst of packets through the new link which may cause severe packet drops leading to retransmission timeouts if the new link's transmission queue does not have the capacity to handle it.

Packet reordering due to a vertical handoff and the conditions for the occurrence of packet reordering in vertical handoff scenarios are discussed in [39,64,125]. In the case of handoff involving links of the same bandwidth, a decrease in the link delay after a handoff generates reordered packets but as the bandwidth remains the same, sufficient number of *dupacks* may not be generated to trigger a false *fast retransmit*. In the case of a handoff between the same delay links, it is not possible to inject *dupthresh* number of

packets through the new link to overtake the packets through the old link. If the ratio of the new access link bandwidth to the old access link bandwidth is greater than *dupthresh*, enough *dupacks* may be generated to trigger a false *fast retransmit* provided the delay of the new link is smaller than that of the old link. An example scenario is shown in Figure 1a, Paper 3. We observe that as the ratio of the old and the new link delays increases, the *fast retransmit* algorithm is triggered immediately after a handoff and the number of unnecessary retransmissions increases. Handoff scenarios where packet reordering occurs are described in detail in Papers 3.

Unused Connection Time and Reduction in *ssthresh*

A break-before-make handoff is likely to result in unused connection time. If the disconnection time is more than the RTO value of a TCP connection, the RTO timer may expire several times during the disconnection period, each time doubling the RTO value [113]. When the connectivity is resumed, the TCP sender needs to wait until the RTO timer expires again before attempting another retransmission. This unused connection time delays the start of the recovery of lost packets. This problem is described in detail in [137].

Our experiments in Paper 1 and Paper 2 show that in addition to the unused connection time, reduction in *ssthresh* due to the occurrence of more than one RTO is a key factor in affecting the TCP performance in a break-before-make handoff. If more than one RTO has occurred, i.e., a retransmission is lost, then the *ssthresh* value is further reduced and the recovery of the lost packets is carried out mostly in the congestion avoidance phase with a very low *ssthresh* value. However, in a break-before-make handoff, the retransmission is lost due to disconnection and not due to congestion and so making this reduction in *ssthresh* is unnecessary. An example scenario is shown in Figure 9a, Paper 1. When the BDP of the new link is greater than the reduced *ssthresh*, TCP performance degrades due to the underutilization of the new path.

Packet Losses

In a vertical handoff, packet losses occur due to (i) a decrease in BDP after a make-before-break handoff, (ii) a decrease in bandwidth after a handoff even if the handoff occurs between same BDP links, and (iii) disconnection due to a break-before-make handoff. The packet losses due to a decrease in BDP and disconnection are studied in [40, 43, 55, 56, 59, 64, 87, 102, 149].

With a make-before-break handoff from a high-BDP link to a low-BDP

link, congestion-related packet losses occur due to the decrease in BDP. A decrease in BDP occurs in the case of a handoff from a high-bandwidth link to a low-bandwidth link where the delay of both the links involved in the handoff remains the same or the new link has a smaller delay compared to the old link. As the TCP sender continues to inject packets into the new link at the same high rate as before in the old link, it may result in buffer overflow of the new link and consequent packet losses.

In the case of same bandwidth links, a make-before-break handoff from a high-delay link to a low-delay link causes packet losses due to a decrease in BDP. Multiple RTOs are required for loss recovery if there is a significant reduction in BDP after a handoff as shown in Figure 20a, Paper 2.

An interesting observation from our experiments is that packet losses can occur in a make-before-break handoff between the same BDP links. Packet losses occur as the TCP sender continues to send packets into the new link at the same rate as that in the old link. Typically, if the ratio of the bandwidths of the old link to the new link is greater than eight, severe packet losses occur. Figure 24a, Paper 2, illustrates a make-before-break handoff from a 1600 Kbps/37 ms link to a 200 Kbps/300 ms link. Here both the links have the same BDP but still packet losses occur. In the handoff between the same BDP access links, when the bandwidth of a link decreases, its delay proportionately increases. Both the decrease in bandwidth and the increase in delay, in effect, increase the delay of the link which causes spurious RTOs to occur. So packet losses occur both due to spurious RTOs and due to TCP continuing to send the packets at the high rate of the old link even after the handoff. Another interesting observation is that even if the BDP of the new link is higher than that of the old link, packet losses can occur if the bandwidth of the old link is significantly greater than that of the old link. In the case of a make-before-break handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link which is reported in Paper 4, the BDPs of the old and new links are 38 and 50 packets respectively. Here also packet losses occur as TCP continues to inject packets at the high rate of the old link even after the handoff and due to the occurrence of spurious RTOs as well.

With a break-before-make handoff, the connectivity that is lost for some duration resumes after the completion of the handoff. This disconnection causes packet losses. An example scenario is shown in Figure 14a, Paper 2.

Slow convergence of RTO

In a handoff from a high-delay link to a low-delay link, the RTO value may be very high compared to the new end-to-end RTT. This problem was first

described in [88]. Our experiments in Paper 2 show that queuing delay in the old link inflates the RTT and RTO values if the old link has a high BDP.

As described in RFC 2988 [113], the equations 3.1, 3.2 and 3.3 show how RTT and RTO values are updated when an acknowledgement for a segment arrives at the TCP sender.

$$RTTVAR = (1 - \beta) * RTTVAR + \beta * |SRTT - R| \tag{3.1}$$

$$SRTT = (1 - \alpha) * SRTT + \alpha * R \tag{3.2}$$

$$RTO = SRTT + max(G, K * RTTVAR) \tag{3.3}$$

where R is the current round-trip time (RTT), SRTT is the smoothed round-trip time and $RTTVAR$ is the round-trip time variation. The parameters α and β are 1/8 and 1/4 respectively. G is the clock granularity and K is a constant that has a value 4. RFC 2988 recommends a minimum RTO value (*minrto*) of 1 second but many implementations including Linux TCP allow a minimum limit of 200 ms for the RTO.

We can see from these equations that the exponential moving average (EWMA) algorithm to update RTT values gives a smaller weight to the current RTT sample and also to the RTT variance of the current sample compared to their respective past values. So after a handoff, the RTO will converge to the RTT of the new low-delay path quite slowly. This convergence is exceptionally slow when the RTT variables are updated only once in an RTT [113]. Another problem with the RTT estimator is that when the RTT suddenly decreases, $RTTVAR$ increases resulting in an overestimated RTO. In some implementations such as Linux TCP, this overestimation problem has been mitigated by introducing a variable, $MDEV$ (mean deviance), equal to $|SRTT - R|$ to calculate the final $RTTVAR$ [134]. If the current RTT value is less than $SRTT$, the current $MDEV$ is given only a weight of 1/32. But in Linux TCP also the RTO estimator starts to decrease slowly to a low value if the RTT values remains at a low level as in the case of a handoff to a low-delay access link [134].

The high RTO value delays the timeout recovery unnecessarily if an RTO recovery is needed relatively soon after a handoff. Our experiments show that slow RTO convergence delays the packet recovery especially in the case of a make-before-break and a break-before-make handoff from a high-BDP, high-delay link to a low-BDP, low-delay link. Here RTO recovery is usually needed due to a large number of packet losses. But the loss recovery is delayed as the RTO value used will be based on the old link and

the convergence to the new low RTO value is slow. An example scenario is shown in Figure 20a, Paper 2.

Inability to Adapt to the Increased Capacity

A vertical handoff to an increased BDP path results in the underutilization of the available capacity due to the inherent inability of TCP to adapt to the high BDP available after a handoff. If the handoff occurs during the congestion avoidance phase, increasing the *cwnd* by one segment in one RTT will take several RTTs for TCP to fully utilize the increased new link capacity. This problem is addressed in [56, 102, 133, 149].

In a wireless overlay networks, with the increase in the bandwidth of an access networks, its delay decreases. For an increase in BDP after a handoff, either the bandwidth or the delay of the new link should be greater than the corresponding value of the old link. If the new access link has a significantly higher bandwidth than that of the old link, TCP may not be able to use the high bandwidth available after the handoff. Problems due to packet reordering can arise in this scenario which may lead to a reduction in *cwnd*. In the other case, as the bandwidth of the new link is less than that of the old link, packet losses and the recovery may affect the effective utilization of the new high capacity available after a handoff. So we observe that in wireless overlay networks, TCP's behaviour is overshadowed by the old link characteristics in the case of a handoff to a link with a higher capacity.

3.4 TCP behaviour with Vertical Handoff: Example Scenarios

To observe the effects of the problems of TCP due to a vertical handoff, we describe some example handoff scenarios where both the bandwidth and delay of the access links change while the BDP of the two links remains the same. This scenario brings out most clearly the combined effect of the changes in bandwidth and delay on TCP behaviour in handoff. For a detailed description of the experiments involving various handoff scenarios and the observations based on them, the reader is referred to the Papers 1, 2 and 3.

We consider a set of experiments in which the bandwidth and delay of one of the links involved in the handoff are varied, while the bandwidth and delay of the other link are kept fixed. We have two sets of fixed bandwidth/delay values for the access links, namely, 200 Kbps/300 ms and

6400 Kbps/9 ms. For the varying link, the bandwidth/delay combinations are 200 Kbps/300 ms, 400 Kbps/150 ms, 800 Kbps/75 ms, 1600 Kbps/37 ms, 3200 Kbps/18 ms and 6400 Kbps/9 ms. With these combinations, both the old and the new access links have a BDP of 10 packets. We perform the experiments in a handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link and vice versa.

Handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link

Figure 3.2 shows that a significant decrease in bandwidth and increase in delay due to a handoff increases the transfer time for make-before-break handoffs. Here we fix the new link bandwidth/delay at 200 Kbps/300 ms while varying the bandwidth and delay of the old link. When there is a significant increase in delay after a make-before-break handoff, TCP suffers from spurious RTOs. Spurious RTOs occur in more than 90% of the handoff points when the ratio of the change in delay is at least eight.

Even when the BDP of the two access links remains the same we have observed that packet losses will occur when there is a significant reduction in bandwidth after a handoff. When the old link bandwidth is 1600 Kbps or higher, (the ratio of the bandwidth of the old link to that of the new link is eight or more) many packets are lost in a make-before-break handoff due to the bursty transmission caused by the arrival of late ACKs triggered at high packet arrival rate over the of the old link resulting in a heavy congestion on the new low-bandwidth link. In most of the cases, recovery needs one or more RTOs and the reduction in the sending rate is drastically affected by the reduced *ssthresh* and *cwnd*. The spurious RTOs and packet losses due to the reduction in bandwidth adversely affect the transfer time after a handoff even when a make-before-break handoff has no packet losses. The resulting increase in transfer time for TCP in a make-before-break handoff is shown in the lower graph in Figure 3.2. The transfer time is almost doubled for a handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link compared to the case of the handoff from a 200 Kbps/300 ms link to another access link with the same bandwidth and delay.

The upper graph in Figure 3.2 shows the transfer time for TCP to transfer 100 packets after a break-before-make handoff. Packet loss due to disconnection is the only problem affecting the handoff to a 200 Kbps/300 ms link either from a 200 Kbps/300 ms link or from a 400 Kbps/150 ms link. In these two cases, as the RTO is higher than the disconnection period, the timeout occurs after the disconnection. When the old link delay is 75 ms or less, timeout occurs during the disconnection period and the retransmitted

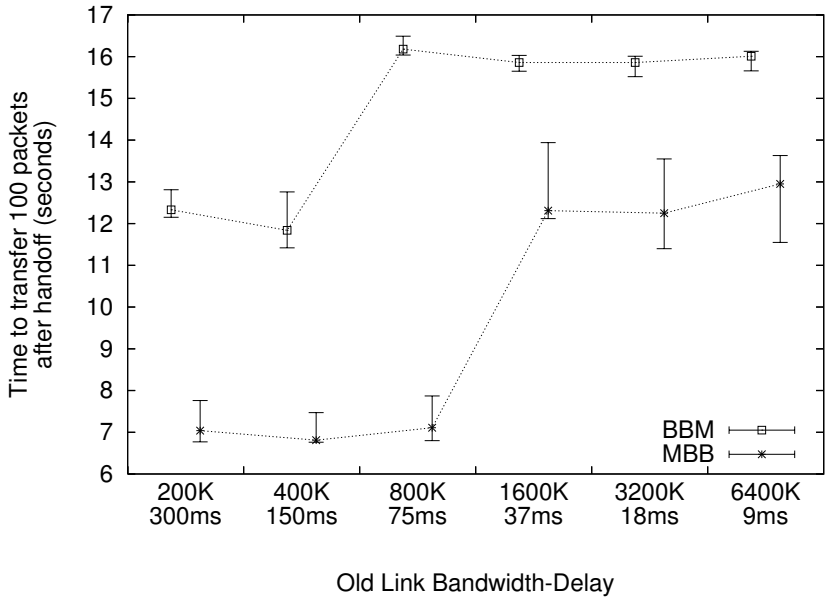


Figure 3.2: Handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link with fixed BDP. Transfer time for 100 packets (y-axis) after a make-before-break (MBB) handoff and a break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link (x-axis) and the bandwidth and delay of the new link fixed at 200 Kbps/300 ms [Figure 6, Paper 2].

packet is also lost. Another timeout is needed to recover the lost packets. So the unused connection time and the *ssthresh* reduction increases the recovery of the lost packets, resulting in an increased time for the transfer of 100 packets after the handoff.

Handoff from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link

Figure 3.3 shows the transfer time for 100 packets after make-before-break and break-before-make handoffs from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link when the bandwidth/delay of the new link is fixed at 6400 Kbps/9 ms. Packet reordering is a problem affecting TCP when there is a significant reduction in delay and a significant increase in bandwidth after a make-before-break handoff. Packet reordering leading to a false *fast retransmit* results in unnecessary reduction in *cwnd* and unnecessary retransmissions. From the lower graph in Figure 3.3 we see

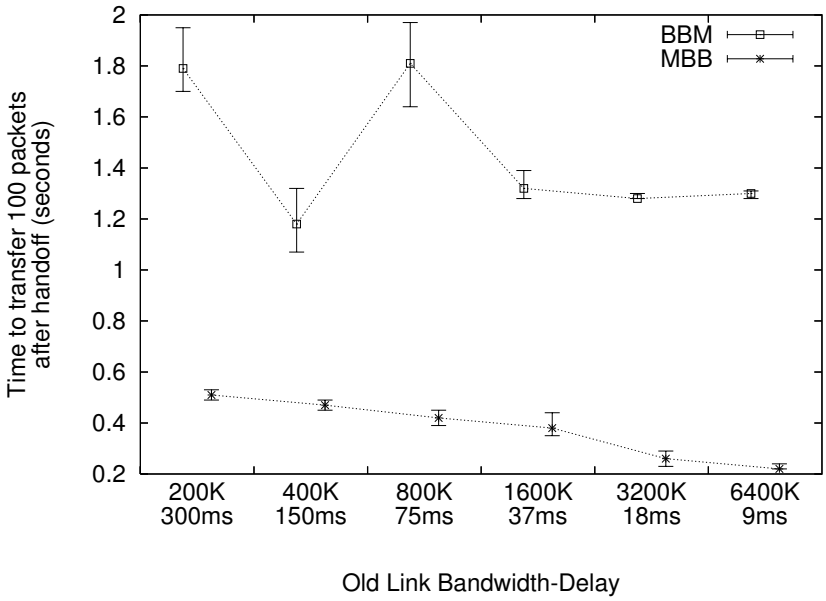


Figure 3.3: Handoff from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link with fixed BDP. Transfer time for 100 packets (y-axis) after a make-before-break (MBB) and a break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link (x-axis) and the bandwidth and delay of the new link fixed at 6400 Kbps/9 ms [Figure 7, Paper 2].

that it takes about 0.2 seconds to transfer 100 packets if the handoff is from a 6400 Kbps/9ms link to a link of identical parameters. But when the handoff is from a 200 Kbps/300 ms link to a 6400 Kbps/9 ms link, it takes 0.6 seconds to transfer 100 packets. This shows the influence of the old link characteristics in determining the transfer time after a handoff.

The upper graph in Figure 3.3 shows the transfer time for 100 packets after a break-before-make handoff to a 6400 Kbps/9 ms link. In the case of a break-before-make handoff from a high-delay old link (e.g., 200 Kbps/300 ms link), the high RTO value prolongs the start of the RTO recovery. The RTO value is around 1.3 seconds when the bandwidth/delay of the old access link is 200 Kbps/300 ms. This means that the RTO timer expires only 800 ms after the disconnection time of 500 ms. For the handoff from a 400 Kbps/150 ms link, the RTO value (around 700 msec) is slightly larger than the disconnection time of 500 ms and the near immediate RTO recovery reduces the transfer time compared to the case of handoff from

a 200 Kbps/300 ms link. In the case of a handoff to a link with a lower delay (75 ms or lower) the increase in transfer time is mainly due to the occurrence of multiple RTOs and the resulting reduction in *ssthresh*. The maximum unused connection time and the reduction in *ssthresh* account for the peak in the transfer time in Figure 3.3 for the break-before-make handoff from a 800 Kbps /75 ms link to a 6400 Kbps/9 ms link.

3.5 Summary

In this chapter we have described our experiments on the problems of TCP in vertical handoff along with related work. We presented a simulation model which includes an accurate modelling of both make-before-break and break-before-make handoffs. We described our experiments involving both make-before-break and break-before-make handoffs to bring out the specific problems of TCP in the different handoff scenarios and to show the extent to which TCP is affected by the changes in bandwidth and delay of the access links involved in a handoff. These experiments enabled us to study the effects of changes in delay and in bandwidth separately and together. The choice of the parameters for the bandwidth and delay cover the entire range of values that are of interest in typical handoff scenarios.

While spurious RTOs are the main problem of TCP with the increase in access link delay after a handoff, packet reordering and slow convergence to the new low RTO value are the problems when a handoff occurs to a low-delay access link. We analyzed the problems of TCP in handoff by identifying the various scenarios that lead to these problems; for example, spurious RTOs occur when there is a significant increase in delay after a handoff. From our experiments we observed that the occurrence of spurious RTOs is dependent on the TCP state as well. Even with a two-fold increase in delay due to a handoff, spurious RTOs can occur due to queueing delay if TCP enters fast recovery after a handoff. The slow convergence to the low RTO value increases the recovery time, especially when packet losses occur due to a large decrease in BDP after a handoff. As the disconnection time of a break-before-make handoff increases, multiple RTOs occur and the unused connection time increases. Another serious problem in this scenario is the repeated reduction in *ssthresh* due to multiple RTOs which allows the recovery only in congestion avoidance and that too with a very small window (usually with size 2). An interesting result of the study is that TCP behaviour is affected due to a vertical handoff between same BDP links. For the same BDP links, packet losses can occur when the bandwidth of the new link is less than eight times the bandwidth of the old link.

The results of our experimental study on TCP behaviour in a vertical handoff reported in Papers 2 and 3 provide a basis for developing the enhanced algorithms presented in Papers 2, 3 and 4 to improve TCP performance in a handoff. Table 3.1 summarizes the various problems of TCP in a vertical handoff along with the scenarios in which they occur.

Table 3.1: Summary of the problems of TCP in a vertical handoff

Problem	**Due to Event**
Spurious RTOs	(i) Make-before-break handoff from a low-delay link to a high-delay link [43, 64, 67, 125] (ii) Make-before-break handoff from a high-bandwidth link to a low-bandwidth link [43]
Packet Reordering leading to false fastretransmit	Make-before-break handoff from a high-delay link to a low-delay link and $bandwidth_{newlink} > dupthresh * bandwidth_{oldlink}$ [39, 64, 125]
Packet Losses	(i) Make-before-break handoff from a high-BDP to a low-BDP link [56, 58, 64, 87, 102, 149] (ii) Make-before-break handoff from a high bandwidth to a low bandwidth link for same BDP links [40, 43] (iii) Disconnection due to a break-before-make handoff [40, 43]
Unused connection time and *ssthresh* reduction	Break-before-make handoff disconnection period greater than the RTO of the old path [40, 43, 137] Repeated RTOs and *ssthresh* reduction [40, 43]
Inability catch up with high capacity	Handoff from a low BDP link to a high BDP link [56, 102, 133, 149]
Slow convergence to the new RTO	Handoff from high-delay link to a low-delay link [40, 43, 88]

Chapter 4

Alleviating the Problems of TCP in Vertical Handoff

In this chapter we describe our cross-layer assisted algorithms to mitigate the problems of TCP in a vertical handoff along with the proposals in the literature. We begin this chapter by reviewing some general TCP algorithms that can be used to deal with the TCP problems in a vertical handoff which do not require any cross-layer information regarding handoff. Section 4.2 describes cross-layer assisted TCP algorithms for vertical handoff with the focus on our proposed algorithms. Section 4.3 gives some example handoff scenarios to illustrate the effectiveness of our proposed algorithms. The interested reader can find a detailed description of the various proposals in the literature along with a comparative discussion of our algorithms in the Papers 1, 2, 3 and 4 that are included in this thesis. Section 4.4 presents a discussion of the proposed algorithms and the Section 4.5 provides a summary of the chapter.

4.1 TCP Algorithms without Cross-layer Information

In this section we give an overview of the TCP algorithms proposed in the literature that can be used to mitigate the problems of spurious RTOs and packet reordering in a general setting. While these algorithms can also be employed to improve the TCP behaviour in vertical handoff scenarios we point out the merits and demerits of such solutions.

DSACK [53] is an extension of the TCP SACK [101] in which the receiver reports to the sender that a duplicate segment has been received. The use of DSACK to detect unnecessary retransmissions either due to

spurious RTOs or due to packet reordering and to undo the unnecessary congestion control actions is described in [25, 26]. When DSACK information is received, the congestion control measures such as the setting of *cwnd* and *ssthresh* that have been taken already are undone (i.e., restoring their previous values) only if all the segments that are retransmitted by the TCP sender in the previous window have been duplicated. In a vertical handoff, TCP may not be able to know for how long it is has to wait to confirm that all the retransmitted packets in a particular window have been unnecessarily transmitted as the characteristics of the access links have changed significantly after a vertical handoff. Besides, the restoration of the old *cwnd* and *ssthresh* may be inappropriate in a vertical handoff scenario as the path characteristics have changed after a vertical handoff.

The TCP-Eifel detection algorithm [97, 98] uses the TCP timestamps option [29] to detect spurious retransmissions. The TCP-Eifel response algorithm [96] describes the methods to undo the unnecessary congestion control measures taken during the RTO recovery. The Eifel detection algorithm provides a faster detection of spurious RTOs compared to DSACK but for every packet there is an overhead of 12 bytes because of the use of timestamps.

Forward RTO-Recovery (F-RTO) [131, 132] is a TCP sender-only algorithm that helps to detect spurious RTOs. Unlike TCP-Eifel it does not require any TCP options to operate. The F-RTO algorithm retransmits the first unacknowledged segment as a response to an RTO. By monitoring the incoming acknowledgements, the algorithm determines whether or not the timeout was spurious and decides whether to send new segments or to retransmit unacknowledged segments.

Although both TCP Eifel and F-RTO are effective in detecting unnecessary retransmission timeouts, the selection of a proper response is hard without additional information about the new path.

Various techniques to increase the *dupthresh* values to avoid the triggering of the *fast retransmit* algorithm due to packet reordering have been proposed in [25]. These techniques use a variant of the *Limited Transmit algorithm* [13] to preserve the ACK-clocking by sending new data for every second *dupack*.

Reordering Robust-TCP (RR-TCP) proposed in [159] uses the DSACK information to vary the *dupthresh* value adaptively for triggering the *fast retransmit* algorithm. It proposes several algorithms for avoiding the false retransmits proactively.

TCP-NCR [24] roughly increases the *dupthresh* value based on the congestion window of data. TCP-NCR also extends TCP's *Limited*

Transmit algorithm to allow the sending of new data during the period when the TCP sender is engaged in distinguishing between loss and reordering.

The above schemes try to avoid triggering false *fastretransmits* due to packet reordering, but they do not take into account the characteristics of the new link in setting the *cwnd* and *ssthresh* after a vertical handoff.

4.2 Cross-layer Assisted TCP Algorithms

TCP congestion control algorithms have been designed to enable TCP to adapt to the fluctuating bandwidth available on its end-to-end path. However, due to the abrupt significant changes in bandwidth and/or delay of the access links caused by a vertical handoff, TCP may experience several problems due to loss of packets, unnecessary retransmissions, packet reordering, spurious timeouts before it can adapt to the new end-to-end path as described in Section 3.3. The assumption that the wireless access link is the bottleneck link of the end-to-end path of a TCP connection is justifiable as the bandwidth of the wired links in the end-to-end path is usually much higher than that of the wireless access links at the end points. So by providing the easy-to-obtain information about the bandwidth and/or delay changes of the access links to TCP it is reasonable to explore whether TCP performance in a handoff can be significantly improved in a majority of the vertical handoff scenarios. Our experiments with cross-layer assisted TCP algorithms which are described in this section provide an affirmative answer.

The Rationale behind Our Algorithms

Previous studies have shown that the end-to-end path of a TCP connection remains fairly stable over the lifetime of a connection [112]. This fact about the stability of TCP routes enables us to make a reasonable assumption that any change in the end-to-end path characteristics that affects TCP behaviour in a vertical handoff is very likely caused by the changes in the access link characteristics due to a handoff.

It is possible for a mobile node to easily obtain the information regarding the occurrence of a vertical handoff and the status of the wireless link: for instance, the mobile node can consider IEEE 802.21 Media Independent Handover (MIH) services [76] as a source for this information. IEEE 802.21 aims at developing mechanisms that provide information regarding the link status and link parameters to the upper layers to optimize the handovers among heterogeneous access networks. For example, the standard

can provide event notifications such as link-up, link-down and link quality is degrading etc., and also information regarding the link level characteristics to the upper layers.

The cross-layer assisted enhancements to the TCP algorithm that we propose to mitigate the problems of TCP in a vertical handoff require modifications only to the TCP sender algorithm. If the TCP sender happens to be on a mobile node, it is easy for the mobile node to inform the changes in the access link characteristics by using local cross-layer notifications. If the TCP sender happens to be at the correspondent node, an explicit end-to-end cross-layer notification from the mobile node will help the TCP sender to get this information. A study on the feasibility of delivering the link information using the mobility protocols shows that many mobility protocols can support this exchange of information [92]. For example, it is possible to send this information along with the binding update messages in Mobile IPv6 [84] or as a part of the *HIP UPDATE* packet with HIP [120]. It is also possible to send this information as a part of the TCP options. In our algorithms we model that the delivery of the handoff notifications is piggybacked in the mobility signalling messages so that they can be delivered to the TCP layer exactly when the handoff completes.

Our proposed enhancements are implemented in the TCP SACK algorithm and they are invoked when a cross-layer notification arrives from the mobile node to the TCP sender at the correspondent node. This cross-layer notification includes information about the occurrence of a handoff and a rough estimate of the bandwidth and delay of the old and the new access links. Our algorithms use these values to check the possibility of the occurrence of various problems due to a vertical handoff and devise bounds for the TCP parameters such as *minrto*, *ssthresh* and *cwnd* to mitigate these problems. Modifications to the TCP sender are also done when all the packets sent before handoff are acknowledged and an ACK for new data is received. The cross-layer notification arrives when the handoff completes at the network layer. The arrival of the ACK for all the packets sent before handoff 'completes' the handoff event at the transport layer. When the ACK for the new data arrives, we can be sure that the data and the ACK have taken the new path after the handoff.

Our algorithms are incremental in nature as we have refined them based on the extent of available cross-layer information. Starting from the case where the available information was whether a handoff occurred and whether there was a significant change in the access link characteristics, we have further developed the algorithms for the case where the available information provides a rough estimate of the bandwidth and delay of the

access links involved in a handoff. To deal with the case of multiple TCP flows, the available information includes the number of TCP flows present. We have developed the various algorithms individually by isolating the different problems of TCP in a handoff. Our experiments in various handoff scenarios cause the various problems of TCP in a handoff to occur together and so our experimental results are based on the combined working of the various algorithms we have designed.

Our algorithms are conservative in the sense that they are designed not to be counter-productive in any situation. Our algorithms are relatively simple and are easy to implement. We conducted experiments based on the implementation of the algorithms in Linux kernel version 2.6.18 and our results show that the performance of the proposed algorithms is quite close to the results obtained in the simulation experiments [83]. In the absence of the cross-layer information, the proposed enhancements do not affect the normal behaviour of the TCP algorithm. We evaluated the proposed algorithms in various handoff scenarios to show their effectiveness. We next describe the key ideas underlying our proposed algorithms often suppressing details that may distract in understanding the main ideas behind them. The complete algorithms are given in the Papers 2, 3 and 4.

Algorithm to Avoid Spurious RTOs

Spurious RTOs occur in a make-before-break handoff from a low-delay link to a high-delay link as the RTO timer at the TCP sender expires before the ACK arrives through the high-delay link after a handoff.

Figure 4.1 shows the key steps of the algorithm to avoid spurious RTOs. In order to avoid the expiry of the RTO timer, we set the RTO based on the RTT of the new high-delay link. Using the bandwidth and delay values of the old and the new access links, we find the traversal time for a *Data-ACK pair* at the time of handoff. If the traversal time is greater than the current RTO, there is a possibility of the occurrence of spurious RTOs and we calculate the *minrto* based on the new access link delay and update the RTO timer immediately so that the new *minrto* comes into effect. For example in the case of handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link, the RTO value at the time of handoff is around 200 ms and the time taken for for the ACK to arrive through the new link alone is 300 ms thereby meeting the condition for the occurrence of spurious RTOs. Our experiments show that the occurrence of spurious RTOs is dependent on the TCP state as well. Even with a two-fold increase in delay due to a handoff, spurious RTOs can occur due to queueing delay if TCP is in fast recovery state. So in the calculation of *minrto* we take into account

When handoff notification arrives
Calculate the traversal time t_{Data_ACK}
for a data packet sent just before handoff
If ($t_{Data\_ACK} > current\_RTO$ OR
TCP already in Fast Recovery)/* Possibility of a Spurious RTO*/
Save *minrto*
Calculate the *minrto* based on the RTT of the new access link
and the FlightSize at the time of handoff
Update RTO timer
When ACK for a new data segment arrives
If there is an increase in access link RTT after a handoff
Initialize and update RTT variables as for a new connection
Restore *minrto*

Figure 4.1: Algorithm to avoid spurious RTOs

the effect of the *FlightSize* along with the delay of the new link. Here *minrto* refers to the minimum bound for the RTO value. As we are only setting the *minrto*, any change in the delay of the end-to-end path will be better reflected in the RTO calculation. When an ACK for a new packet arrives, RTT variables are initialized according to RFC 2988 [113]. This helps the TCP algorithm to converge to the new RTO value quickly. The complete algorithm is given in Figure 11, Paper 2 and it does not require any modification when there are multiple TCP flows.

Paper 1 gives the *incRTO* algorithm to avoid spurious RTOs. It updates the RTO value to 3 seconds, which is the initial RTO value for a new connection when a handoff notification arrives. The RTO is set to this value since the cross-layer notification informs the TCP sender that a handoff has occurred to a new link with a significant increase in delay. In the *RTT inflation scheme* which has been proposed in [124], RTT is increased to an estimated RTT value using ICMP messages and RTT is set to 3 seconds if the ICMP method of estimation fails. The proposal in [67] to avoid spurious RTOs is to reduce the difference in RTT between the old path and the new path so that TCP can gradually adapt to the RTO of the new path.

Algorithm to Combat the Effects of Packet Reordering

Figure 4.2 captures the key steps of our algorithm to combat the effects of packet reordering due to a vertical handoff. A necessary condition for packet reordering leading to *fast retransmit* is the case when the ratio of the bandwidth of the new and the old access links is greater than *dupthresh*.

When handoff notification arrives
If ($BW_{newlink} > 3 \cdot BW_{oldlink}$)
/* Possibility of reordering leading to a fast retransmit */
Calculate $dupthresh = max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$
In Fast retransmit:
Retransmit the first unacknowledged segment
In Fast Recovery:
Transmit a new packet for each *dupack* until all packets sent before handoff are acknowledged or *dupacks* exceed *dupthresh*
If (# of *dupacks* > *dupthresh*)
Return to the normal fast recovery
When ACK for a new data segment arrives
If (DSACK indicates that the retransmission was unnecessary)
Set *cwnd* and *ssthresh* based on $BDP_{newlink}$

Figure 4.2: Algorithm to combat the effects of packet reordering

If this condition is met when the handoff notification arrives, our algorithm calculates a new *dupthresh* value based on the bandwidth of the old and the new access links. If *fast retransmit* occurs, the algorithm retransmits the first unacknowledged packet and in *fast recovery*, it send a new packet for every successive *dupack* thereby utilizing the high bandwidth of the new link. If the number of *dupacks* exceeds the *dupthresh* value, the algorithm goes back to the normal fast recovery. If the retransmission is identified as unnecessary using DSACK and if TCP has not returned to the normal fast recovery, our algorithm infers that the *dupacks* are generated by packet reordering and are not due to congestion, and sets the *cwnd* and the *ssthresh* based on the BDP of the new link scaled by the number of flows. If the condition is not met, our algorithm uses the standard SACK recovery. The algorithm for a single flow is given in Figure 3, Paper 3 and the multiple flow version of this algorithm is given in Figure 3, Paper 4.

A *nodupack* scheme proposed in [64] suppresses the transmission of *dupacks* during a handoff. This may lead to RTO recovery if *dupacks* generated are due to packet losses. An *RTT equalization scheme* is proposed in [124] where RTTs of the packets are equalized by sending the ACK through a fast link if the data packet has arrived through a slow link and vice versa to avoid the generation of *dupacks*. A problem with these schemes is that it may not be able to utilize the high bandwidth available after a handoff even though they can avoid the occurrence of a false *fast retransmit*.

When handoff notification arrives
If (TCP is not in RTO recovery)
If (*FlightSize* > Buffering capacity of the newlink)
Set *cwnd* and *ssthresh* based on $BDP_{newlink}$

Figure 4.3: Algorithm to reduce congestion-related packet losses.

Algorithm to Minimize the Packet Losses

Congestion-related packet losses occur in a vertical handoff if the new path after a handoff has less capacity than the old path. As the access links are most often the bottleneck links, packet losses due to congestion may occur if the *FlightSize* at the time of handoff is greater than the buffering capacity of the new link. In our algorithm given in Figure 4.3, we check whether the above condition arises and if so, set the *cwnd* and *ssthresh* based on the BDP of the new link scaled by the number of flows. Figure 25, Paper 2, gives the algorithm to minimize the packet losses for a single flow and the multiple flow version of this algorithm is given in Figure 1, Paper 4.

The algorithms proposed in [102,149] minimize packet losses by measuring the bandwidth and RTT of the end-to-end path and setting the *cwnd* to their product (i.e., BDP). These measurements may not be accurate as they are based on the arrival of ACKs since both the ACKs and packets may flow through both the old and the new access links in a make-before-break handoff. In [59], overbuffering is proposed to improve the handoff performance of TCP and TFRC in which all the link buffers are to set to the maximum of the BDP of all the links in the end-to-end path. Though overbuffering may help TCP to have a smooth handoff between links of different BDPs, this scheme is not practical as it requires the knowledge of the bandwidth and delay of all the links in the path beforehand.

Algorithm to Reduce the Effects of Disconnection

The unused connection time and the repeated reduction in *ssthresh* are the major problems of TCP arising from the disconnection caused by a break-before-make handoff. Our algorithm to reduce the unused connection time given in Figure 4.4. Here the TCP sender immediately retransmits the first unacknowledged packet if it is already in RTO recovery when the handoff notification arrives. Also the *ssthresh* variable is set to the BDP of the new link to avoid the repeated reduction in *ssthresh*. This algorithm does not alter the value of *cwnd* after an RTO as it is not certain that the RTO has occurred due to a break-before-make handoff. In the multiple flow version

When handoff notification arrives:
If (TCP is in RTO recovery)
Retransmit the first unacknowledged packet
Set *ssthresh* to BDP of the new link
When ACK for a new data segment arrives
If there is an increase in access link RTT after a handoff
Initialize and update RTT variables as for a new connection

Figure 4.4: Algorithm to reduce the unused connection time and to set *ssthresh*

When handoff notification arrives
If the $RTT_{newlink} << RTT_{oldlink}$
Initialize RTT variables as for a new connection
When ACK for a new data segment arrives
Update RTT variables

Figure 4.5: Algorithm for fast convergence of RTO

of this algorithm given in Figure 2, Paper 4, the *ssthresh* is set to the BDP of the new link scaled by the number of flows.

The immediate retransmission of the first unacknowledged segment proposed in [137, 138] addresses only the problem of unused connection time. RFC 5681, "TCP Congestion Control", recommends that the value of *ssthresh* should be reduced only once if the RTO occurs more than once for the same segment. We have independently arrived at the same conclusion as elaborated in Section 3.3 of Paper 1 and in the algorithm given in Figure 4, Paper 1. The algorithm suggested in RFC 5681 avoids the unnecessary reduction of *ssthresh* but in a handoff scenario where the new link has a higher BDP than that of the old link, this value of *ssthresh* is still less than the capacity of the new link and TCP is not able to readily utilize the high BDP.

Algorithm for a Fast Convergence of RTO

Our algorithm for a fast convergence of RTO in the case of vertical handoff to a significantly lower delay link is given in Figure 4.5. When a handoff notification arrives, if there is a significant reduction in the link RTT after a handoff, our algorithm initializes the RTT variables as recommended in RFC 2988. The RTT variables are updated when the acknowledgement for new data arrives. This algorithm is given in Figure 16, Paper 2, and no

modification to this algorithm is needed for multiple TCP flow scenarios.

The proposal in [88] gives three methods to avoid the problems of TCP arising from a sudden change in RTT due to a handoff, namely, the resetting of the TCP retransmission timer after a handoff, obtaining an accurate RTT using timestamps, and increasing the weights β and α in Equations 3.1 and 3.2 for the first few RTO updates immediately after a vertical handoff.

Algorithms to Utilize the High Capacity Available after a Handoff

Our algorithms do not propose any direct method to solve this problem but they solve this problem indirectly for a vertical handoff in the currently available wireless access networks organized as a wireless overlay architecture. In our discussion of this problem in Section 3.3, we have seen that the high BDP scenario after a handoff arises either in a handoff to a new link with a higher bandwidth and lower delay (compared to the old link) or to a link with a lower bandwidth and higher delay. In the above scenarios TCP has problems either due to packet reordering or due to packet losses. As our algorithms given in Figure 4.2 and Figure 4.3 are effective in mitigating the problems of packet reordering and packet losses, they enable an efficient utilization the high capacity available after a handoff.

Scenarios where handoff occurs to a new access link with higher bandwidth and delay than the old link are of interest for study. However, such handoff scenarios do not seem to arise in the present wireless overlay architecture and we have not looked into them. Packet reordering may not occur in these scenarios as the new link delay is higher than that of the old link. Packet losses may not occur as the new link has a higher BDP and bandwidth than the old link. Additional information about the end-to-end path may be required in setting the *cwnd* and *ssthresh* in these scenarios.

A variant of the Quick-Start algorithm [51, 129] that can be applied after a vertical handoff to determine the capacity of the new path is proposed in [133]. An explicit cross-layer handoff notification is employed to trigger the Quick-Start algorithm when the handoff completes. In the original Quick-Start algorithm, only the *cwnd* is set based on the *Quick-Start Response* but after a vertical handoff, *ssthresh* and *cwnd* are made equal to the *Quick-Start Response*. Simulation results given in the paper show that TCP performance in a handoff is improved as Quick-Start is able to estimate the path capacity after a handoff.

To make effective use of the higher BDP of a new path after a handoff, the paper [102] proposes an algorithm in which a mobile node sends multiple partial ACKs to the TCP sender until the transmission rate in-

creases to the bandwidth of the new link. Partial ACKs are acknowledgements which cover new data but not all the data sent so far [52]. However, the sending of more than one ACK for each received packet is not advisable as malicious users can exploit it to increase the sending rate aggressively [17, 135]. Besides the scheme cannot work if TCP byte-counting [12] is used by the TCP sender.

A receiver-based mechanism to address the underutilization problem in a GPRS-WLAN handoff caused by the slow draining of the SGSN buffer is described in [56]. In this proposal, when an impending handoff is detected, the TCP receiver sends the receiver advertised window (*rwnd*) based on the BDP of the new network and this *rwnd* will be effective once the mobility registration is complete. The *rwnd* is increased by two segments so that there will be sufficient *dupacks* to trigger the fast retransmit of the packets queued in the SGSN buffer through the WLAN network. Upon the completion of handoff, *rwnd* can be increased based on the measured BDP of the WLAN network.

A combined TCP receiver-sender approach is proposed in [156] to improve the performance of TCP in a break-before-make handoff. A mobile device uses the cross-layer information to determine the completion of a handoff and to make a rough estimate of the bandwidth of the wireless link. Immediately after the completion of a handoff, the mobile node sends two *dupacks* with a new TCP bandwidth option which contains the estimated bandwidth of the new wireless link. Upon receiving the *dupacks* with the bandwidth option, the TCP sender sets the *ssthresh* to the product of the bandwidth and the smoothed RTT and the *ssthresh* is updated over the next four RTT samples. This scheme may not work well in make-before-break handoff scenarios if the *cwnd* at the time of handoff is greater than the new *ssthresh* value which is calculated from on the bandwidth of the new link. A speedy probing of the correct BDP may not take place as TCP is in the congestion avoidance phase.

TCP Enhancements for Multiple Flows

The cross-layer assisted TCP algorithms we have proposed in Papers 1, 2 and 3 are based on the behaviour of a *single* TCP flow. Using a single TCP flow to study the behaviour of TCP in a vertical handoff and to evaluate the proposed enhancements is not altogether adequate to conclude that the proposed algorithms are effective in mitigating the problems of TCP in a vertical handoff. On the other hand, using a single TCP flow to study the problems of TCP in a handoff is valuable to isolate the problems of TCP without the complexity introduced by the multiple TCP flows. So

as the next step in our study we examine the behaviour of multiple TCP flows when they simultaneously undergo a vertical handoff and also how the algorithms that we have proposed in the case of a single TCP flow can be adapted to multiple TCP flows in a handoff. This work is reported in Paper 4.

Our experiments that are reported in detail in Paper 4 show that all the problems described in Section 3.3 occur in the case of multiple TCP flows in a vertical handoff.

As multiple TCP flows share a bottleneck link, it is no longer valid to assume that the available capacity for a flow after a handoff is equal to the BDP of the new link. We need the capacity of the new link in the following steps in our algorithms: (i) to check for the incipient congestion by comparing the *FlightSize* with the new link BDP in the *algorithm to minimize the packet losses* in Figure 25, Paper 2, (ii) to set the *ssthresh* after a handoff in the *algorithm to reduce the unused connection time and to set the ssthresh* in Figure 15, Paper 2, and (iii) in setting the *cwnd* and *ssthresh* after a handoff in the *algorithm to minimize the effect of packet reordering* in Figure 3, Paper 3. If we set the *cwnd* and *ssthresh* to the BDP of the new link divided by the number of flows sharing the bottleneck link at the time of handoff, this value will represent the share of the capacity of a single flow. Accordingly we model that the mobile node communicates to the TCP sender at the corresponding node the number of simultaneous TCP flows along with the cross-layer notification. The TCP sender then sets the *cwnd* and *ssthresh* after a handoff to the BDP of the new link divided by the number of flows. With this simple modification to the three algorithms stated above, our algorithms are able to adapt to the case of multiple TCP flows in a vertical handoff.

We have evaluated our algorithms for one, two and four TCP flows. As the number of flows increases, the size of *cwnd* decreases inversely proportional to the number of flows. So the effect of problems such as spurious RTOs, packet reordering have a diminishing impact on TCP performance. For instance, if the *cwnd* is small, the number of outstanding packets at the time of handoff will be small. So there will be a small number of packets that are retransmitted unnecessarily due to the occurrence of a spurious RTO. We have observed that in the case of sufficiently large values of *cwnd*, the effect of our algorithms in improving the TCP performance is visible.

We are not aware of any specific previous study dealing explicitly with TCP behaviour in a vertical handoff when many flows are present.

4.3 Effectiveness of Cross layer Algorithms: Example Scenarios

The performance of the algorithms described in Section 4.2 are compared with TCP (the TCP SACK algorithm) in many handoff scenarios. These experiments and their results are described in Papers 2, 3 and 4. The TCP SACK algorithm modified by our cross-layer assisted enhancement is referred to as Enhanced-TCP (which is shown as ETCP in the graphs) and it can be seen to reduce the adverse effects of a handoff considerably in most of the scenarios. In this section we show the effectiveness of Enhanced-TCP in some example handoff scenarios which include those scenarios we have described earlier in Section 3.4 to illustrate the problems of TCP in a vertical handoff. We also present some handoff scenarios to illustrate the case of multiple TCP flows.

Handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link

As described earlier in Section 3.4, the main problems that TCP faces in a make-before-break handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link are the occurrence of spurious RTOs and packet losses due to significant decrease in bandwidth after the handoff. For a break-before-make handoff in the same scenario, the unused connection time, repeated reduction of *ssthresh* and packet losses are the main problems of TCP arising from a disconnection. To illustrate our results we have chosen two example scenarios here. In the first case of a handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link (which is the same scenario as described in Section 3.4), the BDP of the two access links involved in the handoff remains the same. In the second case, the handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link, the BDP of the new link after the handoff is higher than that of the old link. The occurrence of packet losses in both these scenarios shows that packet losses occur not only in the case of a make-before-break handoff to a lower BDP link which is rather well-known, but also in the case where a handoff occurs to a link of the same BDP link or a higher BDP if there is a significant reduction in bandwidth after the handoff.

Figure 4.6 shows the comparison between the time taken by TCP and Enhanced-TCP to transfer 100 packets after both a make-before-break handoff and a break-before-make handoff. The bandwidth and delay of the old link are varied as shown in the figure but the bandwidth and delay

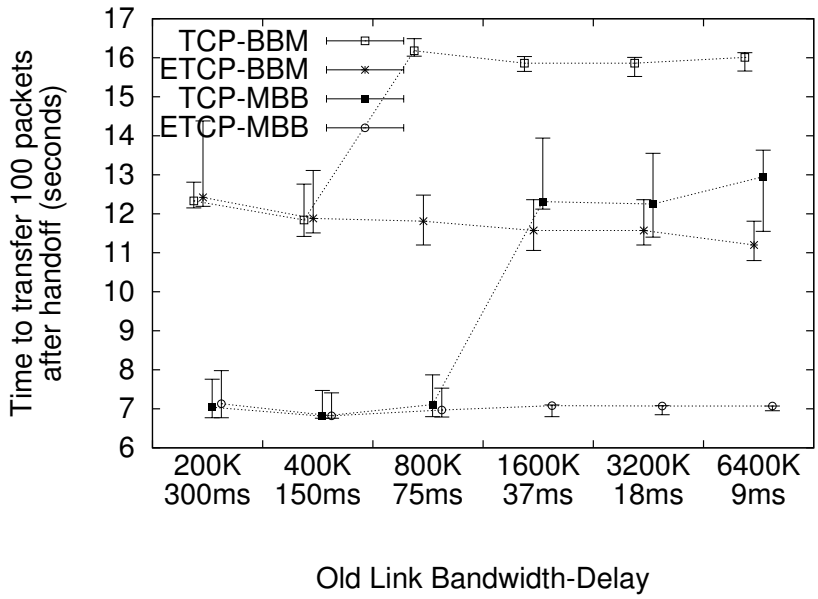


Figure 4.6: Handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link with fixed BDP. Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link and the bandwidth and delay of the new link fixed at 200 Kbps/300 ms [Combines Figure 23a and Figure 26 of Paper 2].

of the new link are fixed at 200 Kbps/300 ms. The BDP of both the old and the new link is 10 packets. There is up to 40 % reduction in transfer time with Enhanced-TCP when the ratio of the bandwidths of the old and the new link is greater than eight as it avoids the occurrence of spurious RTOs in the make-before-break handoff. In the break-before-make handoff case also Enhanced-TCP shows a similar reduction in transfer time by reducing the unused connection time and by setting the *ssthresh* to the correct value of the BDP of the new link. For the Enhanced-TCP, the median value of the transfer time remains the same for all handoff scenarios. This shows that Enhanced-TCP is adapting to the new link characteristics quickly.

Figure 4.7 shows the comparison between the time taken by TCP and Enhanced-TCP to transfer 100 packets after both a make-before-break handoff and a break-before-make handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link. Here the BDP of the old and the new links are 38 packets and 50 packets respectively. There is about 20-40 % reduction

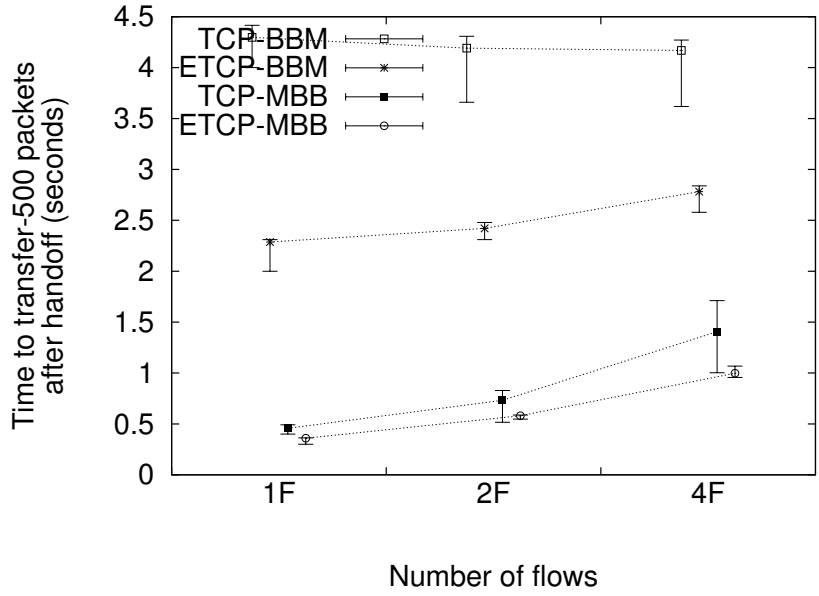


Figure 4.7: Time taken to transfer 100 packets after a make-before-break and break-before-make handoffs from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link [Combines Figure 7 and Figure 11 of Paper 4].

in transfer time for the Enhanced-TCP in the make-before-break handoff compared to TCP. As the links have sufficiently high BDP values, the effectiveness of our algorithm can be seen when there are four simultaneous TCP flows. In the break-before-make handoff scenario also, Enhanced-TCP reduces the transfer time by about 20-40 % compared to TCP.

Handoff from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link

As described earlier in Section 3.4, the main problems in the make-before-break handoff scenario are packet reordering and unnecessary retransmissions. For a break-before-make handoff in the same scenario, the unused connection time, repeated reduction of *ssthresh* and the packet losses are the main problems of TCP arising from a disconnection.

Figure 4.8 shows the comparison between the time taken by TCP and Enhanced-TCP to transfer 100 packets after both a make-before-break handoff and a break-before-make handoff. The bandwidth and delay of the old link are varied as shown in the figure but the new link bandwidth and delay are fixed at 6400 Kbps/9 ms. The BDP of the access links remain

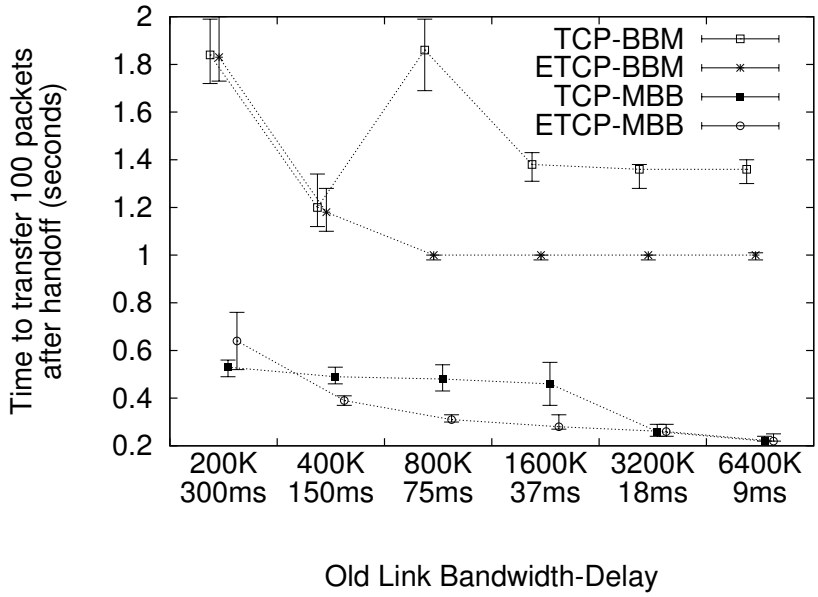


Figure 4.8: Handoff from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link with fixed BDP. Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link and the bandwidth and delay of the new link fixed at 6400 Kbps/9 ms [Figure 27, Paper 2]

the same before and after the handoff. For the make-before-break handoff from a 200 Kbps/300 ms link to a 6400 Kbps/9 ms link, Enhanced-TCP has a slightly higher median value for the transfer time than TCP. This is due to the fact that Enhanced-TCP while utilizing the new path waits for the packets sent before handoff to arrive through the slow 200 Kbps/300 ms link. Enhanced-TCP has a slightly less transfer time as the bandwidth of the old link increases (for the case of 400 Kbps/150 ms, 800 Kbps/75 ms and 1600 Kbps/37 ms links). In the case of a handoff from a 3200 Kbps/18 ms link to a 6400 Kbps/9 ms link, sufficient *dupacks* will not arrive as the ratio of bandwidth of the old link to the new link is less than the *dupthresh* value three. We observe that with Enhanced-TCP there is only one retransmit, the *fastretransmit* packet, while with TCP the median value of unnecessary retransmissions is 10.

For the break-before-make handoff shown in the upper graphs in Figure 4.8, the median value of the transfer time for Enhanced-TCP remains the

same for all handoff scenarios (approximately 1 s) except in the case of the old link values of 200 Kbps/300 ms and 400 Kbps/150 ms. In these two cases the disconnection time of 500 ms is smaller than the RTO value and our immediate retransmission algorithm will not be triggered. Enhanced-TCP can achieve a 20-45 % reduction in transfer time compared to TCP.

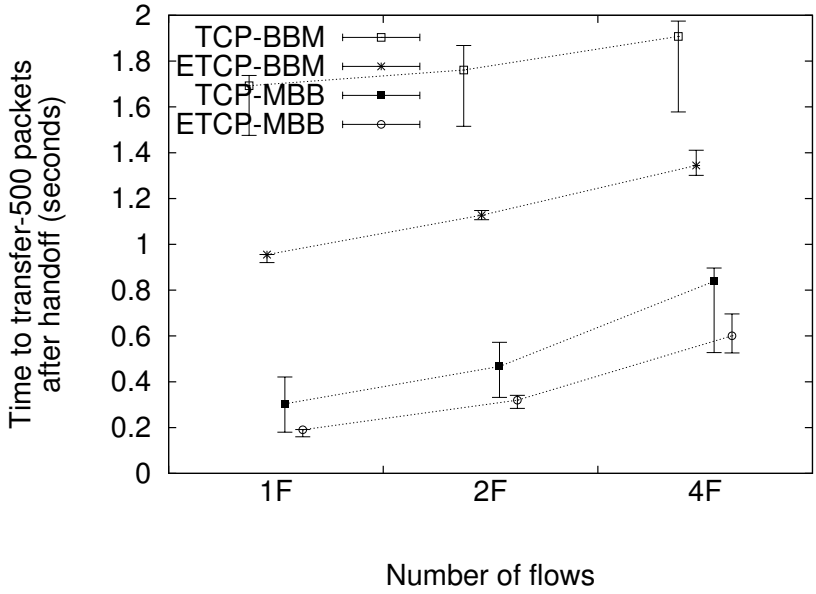


Figure 4.9: Time taken to transfer 500 packets after make-before-break and break-before-make handoffs from a 6000 Kbps/50 ms link to a 54000 Kbps/4 ms link [Combines Figure 15 and Figure 19 of Paper 4].

Figure 4.9 shows a comparison between the time taken by Enhanced-TCP and TCP to transfer 500 packets after both a make-before-break handoff and a break-before-make handoff from a 6000 Kbps/50 ms link to a 54000 Kbps/4 ms link. The BDP of the old and the new links are 50 packets and 38 packets respectively. The lower graphs in Figure 4.9 show the transfer time for 500 packets using Enhanced-TCP and TCP in a make-before-break handoff. Enhanced-TCP takes roughly 30 % less time than TCP as it is able to utilize the high bandwidth link while waiting for the packets to come through the slow link and also avoid unnecessary retransmissions. We observe that in this handoff scenario involving a single TCP flow, TCP unnecessarily retransmits 36 packets (median value) while Enhanced-TCP retransmits only the first unacknowledged packet. The transfer times for the break-before-make handoff shown in the upper

graphs in the same figure indicate that the Enhanced-TCP obtains about 50 % reduction in transfer time compared to TCP in all the cases of one, two and four TCP flows.

4.4 Discussion

Our experimental results show that the cross-layer assisted TCP algorithms enable TCP to adapt faster to the changes in the path characteristics due to a vertical handoff. While the general algorithms such as TCP-Eifel and F-RTO are able to detect the problems of TCP such as spurious RTOs and packet reordering, their responses after the detection are not adequate in vertical handoff scenarios as they do not take into consideration the changes in the path characteristics.

The TCP algorithms that we have developed using the cross-layer information are conservative in nature. In all our algorithms we combine the cross-layer information with the current state of TCP (as reflected in the values of the TCP parameters) to evaluate our hypotheses on the potential problems in a handoff and to set a bound for the TCP congestion control parameters. As a result, our algorithms are robust and they can tolerate small variations in the cross-layer information. In the absence of cross-layer information the proposed algorithms do not adversely influence the TCP behaviour. We next describe how our algorithms use the cross-layer information conservatively.

TCP congestion control algorithms should not attempt to increase the *cwnd* only based on the information about the increased link capacity in the absence of adequate information about the end-to-end path [130]. In accordance with this recommendation, our enhanced TCP algorithms combine the TCP state information with the cross-layer notification to set the *cwnd* and *ssthresh* in a handoff to a link with a higher capacity. For instance, in the algorithm *to avoid the problems of packet reordering* (Figure 3, Paper 4), as packet reordering that triggers a false *fast retransmit* usually occurs when the new link bandwidth is at least *dupthresh* times higher than the old link bandwidth, we use DSACK information also in setting the *cwnd* based on the new access link bandwidth.

Even in the case of a handoff involving same BDP links, packet bursts leading to packet losses can occur when the old link has a much greater bandwidth (more than eight times) than the new link. Our algorithm *to reduce the packet losses* (Figure 1, Paper 4) takes into account the bandwidth and BDP of the access links along with the *FlightSize* at the time of handoff to determine whether packet bursts are likely and suitably sets

the *cwnd* and *ssthresh* to avoid packet losses thereby eliminating expensive packet recovery using the low-bandwidth link.

The algorithm *to avoid the problems of packet reordering* (Figure 3, Paper 4), sends a new packet in response to each *dupack* while waiting for packets to arrive through the old link. This may create a buffering problem if a large number of packets were in transit through the old link at the time of handoff. There will not be a large number of packets in transit as the packet reordering that triggers a false *fast retransmit* usually occurs when the new link bandwidth is at least *dupthresh* times greater than the old link bandwidth. We also check the possibility of congestion in the new link based on the *FlightSize*, BDP and bandwidth of the access links and in the case of an incipient congestion, new packets will not be sent in response to *dupacks*.

The problems of TCP in a handoff to a link of significantly smaller delay have not received adequate attention in the literature. If a significant reduction in BDP occurs due to a handoff to a link with a smaller delay (e.g., Figure 20a, Paper 2), a large number of packets will be lost and usually multiple RTOs are needed for the recovery of the lost packets. As the RTT update algorithm gives less weight to the current sample and the RTO is updated only once in a window, the RTO value will still be high for many RTTs after the handoff. If multiple RTOs are needed for recovery as in the above scenario, this high value of RTO will be doubled every time the RTO recovery fails leading to a very long recovery process. The *algorithm for the fast convergence of RTO* (Figure 16, Paper 2) is triggered in a handoff to a link with a significantly smaller delay and if an RTO recovery is needed the recovery will occur soon as the RTO value is now based on the small RTT after the handoff. In Figure 20b, Paper 2, the *algorithm to reduce the packet losses* eliminates the occurrence of packet losses and no RTO recovery is needed in this scenario.

Our cross-layer notifications do not take into account whether the handoff is a make-before-break handoff or a break-before-make handoff, although the mobile node could get this information. We find in our experiments that this information is unnecessary in most handoff scenarios. However, setting the RTO based on the new link delay to avoid spurious RTOs can increase the recovery time in a break-before-make handoff. In this case, the knowledge that a break-before-make handoff has occurred is helpful and Enhanced-TCP, instead of invoking the *algorithm to avoid spurious RTOs* (Figure 11, Paper 2) which is to be triggered only in a make-before-break handoff, could trigger *the immediate retransmission algorithm* (Figure 2, Paper 4) to speedup the loss recovery. If the disconnection time is greater

than the RTO, this problem will not arise as Enhance-TCP invokes the *immediate retransmission algorithm*.

Our algorithms can be improved by incorporating some methods to quickly estimate the end-to-end bandwidth and RTT after a handoff and using these estimates to set the congestion control parameters. We are currently exploring this topic.

4.5 Summary

This chapter describes our cross-layer assisted solutions to mitigate the problems of TCP in a vertical handoff along with the related work. In Papers 2, 3 and 4 we analyzed the problems of TCP in handoff by identifying the various scenarios that lead to these problems and devised algorithms specific to these scenarios to overcome the problems. We proposed solutions to the problems of spurious RTOs, congestion-related packet losses, prolonged disconnections and slow convergence to the new RTO which arise in a vertical handoff.

Our proposed enhancements are implemented at the TCP sender and they are invoked when a cross-layer notification arrives from the mobile node to the TCP sender at the correspondent node. This cross-layer notification includes information about the occurrence of a handoff and an estimate of the bandwidth and delay of the old and the new access links. Our algorithms use these values to devise a bound for the TCP parameters such as *minrto*, *ssthresh* and *cwnd*. Our algorithms are conservative in nature and experimental results show that they are not counter-productive in any situation. Our algorithms are relatively simple and are easy to implement. In the absence of cross-layer information, TCP behaviour is virtually unaltered.

In our experiments we have evaluated the proposed algorithms in various scenarios to show their effectiveness. Our algorithms can yield up to 40 % reduction in the transfer time immediately after a handoff. Our results show that TCP has severe performance problems even in a handoff between same-BDP links and our algorithms are effective in this scenario.

Table 4.1 provides a summary of the problems of TCP in a vertical handoff along with our proposed solutions.

Table 4.1: Summary of the problems of TCP due to a vertical handoff and the proposed enhancements to the TCP Sender algorithm

Problem	**TCP Sender enhancements**
Spurious RTOs	Set minimum RTO value based on the bandwidth and delay of the new link [Figure 11, Paper 2]
Packet Reordering	If $\frac{BW_{newlink}}{BW_{oldlink}} > dupthresh$, possibility of reordering, Limited transmit, use DSACK to detect unnecessary retransmission set *cwnd* and *ssthresh* based on $BDP_{newlink}$ [Figure 3, Paper 4]
Packet Losses	Set the *cwnd* and *ssthresh* to the BDP of the new link in the case of an incipient congestion [Figure 1, Paper 4]
Unused connection time	If TCP is in RTO recovery, retransmit the first unacknowledged packet immediately and set *ssthresh* based on BDP of the new link [Figure 2, Paper 4]
Slow convergence of RTO	Initialize RTT Variables and update RTO if the handoff is to a lower delay link [Figure 16, Paper 2]
Inability to catch up with high capacity	The algorithm to mitigate problems due to packet reordering and algorithm to avoid packet losses can be used [Figures 1, 3, Paper 4]

Chapter 5

Conclusions and Future work

In this thesis, we have studied the effect of a vertical handoff on TCP. The focus of this research has been to perform a systematic study of the behaviour of TCP in the presence of a vertical handoff and use this study to develop cross-layer assisted TCP algorithms that are implemented at the TCP sender to improve the performance of TCP in a vertical handoff.

The wireless access networks by their very nature effect a tradeoff between geographical coverage, bandwidth and propagation delay. Ubiquitous mobility requires a mobile node to switch between the different wireless networks based on the best of connectivity, application needs, user preferences, etc. The concept of a wireless overlay architecture supports the internetworking of the various wireless access networks by enabling a mobile node to switch seamlessly between these networks. A vertical handoff results in a changeover of the wireless access network that a mobile node uses to connect to the Internet. However 'smooth' the handoff is made to be by the mobility and handoff mechanisms, the abrupt changes in the bandwidth and delay of the wireless links involved in a handoff significantly impact on TCP. TCP on its own can adapt to the changes in the path characteristics at the cost of loss of performance and time for recovery. With the growing popularity of wireless access to a multitude of Internet services and the ever declining cost of the mobile devices equipped with multiple radio interfaces to connect to the Internet, it is expected that a handoff can be a common occurrence in the lifetime of a TCP connection. The wireless access being the endpoints of a TCP connection and also most likely the potential bottlenecks of the end-to-end path, make it possible to explore ways to reduce the adverse effects of the handoff by providing the cross-layer information about the path changes to TCP. We have used the cross-layer information about the access link characteristics to provide a useful hint to the TCP algorithm in adapting to a vertical handoff.

In our simulation study we have explored the effect of handoff on TCP behaviour for a wide range of bandwidth and delay of the access links. Our studies point out the following conclusions regarding the TCP behaviour: (i) a significant change in bandwidth and/ or delay of the access links due to a handoff adversely affects TCP performance, (ii) the performance of TCP after a handoff depends on the state of the TCP when the handoff occurs, (iii) even if the BDP of the access links before and after the handoff remains the same or if the handoff occurs to a higher BDP access link, packet losses will occur if there is a significant decrease in bandwidth of the access link after a handoff, (iv) regardless of how smooth the handoff, TCP may still have performance problems.

We have proposed cross-layer assisted algorithms implemented at the TCP sender to improve TCP performance in a vertical handoff. We use the access link characteristics, such as bandwidth and delay, obtained through the cross-layer notification to put an upper or lower bound on the TCP parameters such as *cwnd*, *ssthresh* and *minrto*. In all our algorithms we use the cross-layer information along with the TCP state at the time of handoff to test our hypotheses regarding the occurrence of a particular problem (e.g., occurrence of spurious RTOs, possibility of packet reordering) and then to set the TCP parameters accordingly. This makes our algorithms conservative and robust. We have shown that our algorithms effectively address the problems of TCP arising from spurious RTOs, packet reordering, packet losses, prolonged disconnection and slow convergence to the new RTO value due to a handoff. The proposed algorithms are simple and are implemented at the TCP sender with no modification to the TCP receiver. These algorithms have been evaluated in both make-before-break and break-before-make handoffs in the ns-2 simulator for access links with a wide range of bandwidth and propagation delay of interest in real-world access networks. Our algorithms are shown to yield significant improvement in TCP performance for many vertical handoff scenarios. These algorithms can be easily adapted for handoff in multiple TCP flows. In the absence of cross-layer notifications, our algorithms have no effect on TCP behaviour. We think that our proposed algorithms are practical and can be adapted for real-world solutions in wireless access network environments.

Moving to a higher bandwidth environment after a vertical handoff is challenging to TCP as TCP needs to be more aggressive to fully utilize the available bandwidth but this cannot be done safely based on the information about the access link characteristics alone. A TCP sender should probe the new network path, but often it takes several RTTs before TCP can adapt to the new path. In the future, we intend to study how TCP can combine the

notifications regarding the access link characteristics with the information gathered by probing the new network path and thereby try to converge quickly and safely to the available network capacity.

It will be of interest to conduct experiments in real access network environments such as EGPRS, UMTS, WLAN and WiMAX along with the mobility protocols such as Mobile IP, HIP and Proxy Mobile IP to evaluate the proposed algorithms for handling both real-time and elastic traffic and to explore ways to estimate quickly and reliably the characteristics of the end-to-end path.

Vertical handoff has been an active research topic for the last ten years or so, yet the level of deployment in practice is not widespread. In the Next Generation Networks (NGN), vertical handoff and seamless mobility are essential elements to achieve the "always best connected" experience. Adapting the research proposals and solutions to the real world presents opportunities and challenges for mobile networking.

References

[1] 3GPP. Enhanced Uplink. Technical Report 3GPP TS 25.321.

[2] 3GPP. Evolved High Speed Packet access (HSPA), Release 7.

[3] 3GPP. High Speed Packet access (HSPA) Release 5 and 6.

[4] 3GPP. Long Term Evolution (LTE), Release 10.

[5] 3GPP. 3rd Generation Partnership Project; Radio Interface Protocol Architecture. Technical Report 3G TS 25.301 version 3.4.0, March 2000.

[6] 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Architecture Principles. Technical Report 3G TR 23.821 version 1.0.1, July 2000.

[7] 3GPP. 3rd Generation Partnership Project; Technical Specification Group Core Network; Restoration Procedures. Technical Report 3G TS 23.007 version 4.0.0 , March 2001.

[8] 3GPP. 3rd Generation Partnership Project; Technical Specification Group Core Network; General Packet Radio Service (GPRS); GPRS Tunneling Protocol (GTP) across the Gn and Gp Interface. Technical Report 3G TS 29.060 version 5.0.1, January 2002.

[9] 3GPP. 3rd Generation Partnership Project; Technical Specification Group Services and Systems Aspects; General Packet Radio Service (GPRS); Service Description; Stage 2; Release 5. Technical Report 3G TS 23.060 version 5.0.0, January 2002.

[10] 3GPP. Feasibility Study on 3GPP System to WLAN Interworking. Technical Report 3GPP TR 22.934 v1.2.0, May 2002.

[11] 3GPP. High Speed Downlink Packet Access (HSDPA); Overall UTRAN description. Technical Report 3GPP TS 25.855, March 2002.

[12] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). Internet RFCs, ISSN 2070-1721, RFC 3465, February 2003.

[13] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. Internet RFCs, ISSN 2070-1721, RFC 3042, January 2001.

[14] M. Allman and A. Falk. On the Effective Evaluation of TCP. *ACM Computer Communication Review*, 5(29), October 1999.

[15] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. Internet RFCs, ISSN 2070-1721, RFC 3390, October 2002.

[16] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. Internet RFCs, ISSN 2070-1721, RFC 5681, September 2009.

[17] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. Internet RFCs, ISSN 2070-1721, RFC 2581, April 1999.

[18] B. Crow and I. Widjaja and J. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, pages 116–126, September 1997.

[19] M. Baker, X. Zhao, S. Cheshire, and J. Stone. Supporting Mobility in MosquitoNet. In *ATEC '96: Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, pages 11–11, 1996.

[20] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4), December 1995.

[21] N. Banerjee, W. Wei, and S. K. Das. Mobility support in Wireless Internet. *IEEE Wireless Communications*, 10(5):54–61, October 2003.

[22] C. Bettstetter, H-Jörg Vögel, and J. Eberspächer. GSM Phase 2+ General Packet Radio Service GPRS: Architecture, Protocols, and Air Interface. *IEEE Communication Surveys*, 2(3), 1999.

[23] P. Bhagwat, S. K. Tripathi, and C. Perkins. Network Layer Mobility: An Architecture and Survey. *IEEE Personal Communications*, 3(3):54–64, 1996.

[24] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton. Improving the Robustness of TCP to Non-Congestion Events. Internet RFCs, ISSN 2070-1721, RFC 4653, April 2006.

[25] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM SIGCOMM Computer Communication Review*, 32(1):20–30, January 2002.

[26] E. Blanton and M. Allman. Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions. Internet RFCs, ISSN 2070-1721, RFC 3708, February 2004.

[27] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP. Internet RFCs, ISSN 2070-1721, RFC 3517, April 2003.

[28] W. Bolton, Y. Xiao, and M. Guizani. IEEE 802.20: Mobile Broadband Wireless Access. *IEEE Wireless Communications*, 14(1):84–95, February 2007.

[29] D. Borman, R. Braden, and V. Jacobson. TCP Extensions for High Performance. Internet RFCs, ISSN 2070-1721, RFC 1323, May 1992.

[30] R. Braden. Requirements for Internet Hosts – Communication Layers. Internet RFCs, ISSN 2070-1721, RFC 1122, October 1989.

[31] G. Brasche and B. Walke. Concepts, Services and Protocols of the New GSM Phase 2+ General Packet Radio Service. *IEEE Communications Magazine*, 35(8):94–104, August 1997.

[32] M. Buddhikot, G. Chandranmenon, S.J. Han, Y.W. Lee, S. Miller, and L. Salgarelli. Integration of 802.11 and Third Generation of Wireless Data Networks. In *IEEE Infocom*, April 2003.

[33] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environment. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, June 1995.

[34] Caida. Analyzing UDP usage in Internet traffic. Available at: http://www.caida.org/research/traffic-analysis/tcpudpratio, 2009.

[35] Caida. Internet Traffic Classification. Available at: http://www.caida.org/research/traffic-analysis/classification-overview, 2009.

[36] R. Chakravorty, P. Vidales, K. Subramanian, I. Pratt, and J. Crowcroft. Performance Issues with Vertical Handovers: Experiences from GPRS Cellular and WLAN Hot-spots Integration. In *Proc. IEEE Pervasive Communications and Computing Conference, (IEEE PerCom 2004)*, March 2004.

[37] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *Proceedings of INET'98*, July 1998.

[38] E. Dahlman, S. Parkvall, J. Skold, and P. Beming. *3G Evolution: HSPA and LTE for Mobile Broadband, 2nd edition*. Academic Press, 2008.

[39] L. Daniel, I. Järvinen, and M. Kojo. Combating Packet Reordering in Vertical Handoff Using Cross-Layer Notifications to TCP. In *Proceedings of the IEEE International Conference on Wireless and Mobile Computing (WiMob'08)*, pages 297–303, October 2008.

[40] L. Daniel and M. Kojo. Adapting TCP for Vertical Handoffs in Wireless Networks. In *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06)*, pages 151–158, November 2006.

[41] L. Daniel and M. Kojo. TCP Behaviour with Changes in Access Link Bandwidth and Delay During Vertical Handoffs. In *Proceedings of the International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST'07), Broadband Wireless Access Workshop*, September 2007.

[42] L. Daniel and M. Kojo. Using Cross-layer Information to Improve TCP performance with Vertical Handoffs. In *Proceedings of the 2nd International Conference on Access Networks (Accessnets'07), Broadband Wireless Internet Workshop*, August 2007.

[43] L. Daniel and M. Kojo. Employing Cross-layer Assisted TCP Algorithms to Improve TCP Performance with Vertical Handoffs. *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, pages 433–465, November 2008.

[44] L. Daniel and M. Kojo. The Performance of Multiple TCP Flows with Vertical Handoff. In *Proceedings of the 7th ACM International Symposium on Mobility Management and Wireless Access (MobiWac'09)*, pages 17–25, October 2009.

[45] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC'07)*, pages 43–56, 2007.

[46] W.M. Eddy. At What Layer Does Mobility Belong? *IEEE Communications Magazine*, pages 155–159, October 2004.

[47] W.M. Eddy and Y. Swami. Adapting End Host Congestion Control for Mobility. Technical Report CR-2005-213838, NASA Glenn Research Center, September 2005.

[48] C. Eklund, R.B Marks, K.L Stanwood, and S. Wang. IEEE Standard 802.16: A Technical Overview of the Wireless MAN Air Interface for Broadband Wireless Access. *IEEE Communications Magazine*, 40:98–107, June 2002.

[49] K. El Malki (ed.). Low Latency Handoffs in Mobile IPv4. Internet RFCs, ISSN 2070-1721, RFC 4881, June 2007.

[50] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26(3):5–21, July 1996.

[51] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-Start for TCP and IP. Internet RFCs, ISSN 2070-1721, RFC 4782, January 2007.

[52] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. Internet RFCs, ISSN 2070-1721, RFC 3782, April 2004.

[53] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgment (SACK) Option for TCP. Internet RFCs, ISSN 2070-1721, RFC 2883, July 2000.

[54] A. Ford, C. Raiciu, M. Handley, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. Internet draft "draft-ietf-mptcp-architecture-02.txt", October 2010. Work in progress.

[55] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta. Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM) (3)*, pages 1537–1545, 2000.

[56] Y. Gou, D.A.J. Pearce, and P.D. Mitchell. A Receiver-based Vertical Handover Mechanism for TCP Congestion Control. *IEEE Transactions on Wireless Communications*, 5(10):2824–2833, October 2006.

[57] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil. Proxy Mobile IPv6. Internet RFCs, ISSN 2070-1721, RFC 5213, August 2008.

[58] A. Gurtov and F. Floyd. Modeling Wireless Links for Transport Protocols. *ACM Mobile Computing and Communications Review*, 34(2):85–96, April 2004.

[59] A. Gurtov and J. Korhonen. Effect of Vertical Handovers on Performance of TCP-Friendly Rate Control. *ACM Mobile Computing and Communications Review*, 8(3):73–87, July 2004.

[60] A. Gurtov and R. Ludwig. Evaluating the Eifel Algorithm for TCP in a GPRS Network. In *Proceedings of European Wireless '02*, February 2002.

[61] H. Holma and A. Toskala. *WCDMA for UMTS - Radio Access for Third Generation Mobile Communications*. John Wiley & Sons, 2000.

[62] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley. Multi-path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet. *IEEE/ACM Transactions on Networking*, 14(6), 2006.

[63] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. Internet RFCs, ISSN 2070-1721, RFC 2543, March 1999.

[64] W. Hansmann and M. Frank. On Things to Happen During a TCP Handover. In *Proc. 28th IEEE Conference on Local Computer Networks (LCN'03)*, pages 109– 118, October 2003.

[65] J. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *ACM SIGCOMM*, August 1996.

[66] M. Honkanen, K. Hugl, J. Ollikainen, A. Parssinen, and C. Wijting. Nokia Radio Research in GIGA. GIGA Results Promotion 2009, Helsinki, GIGA - Converging Networks 2005-2010, October 2009.

[67] H. Huang and J. Cai. Improving TCP Performance during Soft Vertical Handoff. In *Proc. 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, volume 2, pages 329–332, March 2005.

[68] IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-Speed Physical Layer in the 5 GHz band. IEEE Std 802.11a-1999, 1999.

[69] IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band. IEEE Std 802.11b-1999, 1999.

[70] IEEE. IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems. IEEE 802.16-2001, April 2002.

[71] IEEE. IEEE 802.20 Requirements Document Version 9, November 2003.

[72] IEEE. IEEE standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. ANSI/IEEE Std 802.11. 1999 Edition (R 2003), 2003.

[73] IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Further Higher Data Rate Extension in the 2.4 GHz Band. IEEE Std 802.11g-2003, 2003.

[74] IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Spectrum and Transmit Power Management Extensions in the 5 GHz Band in Europe. IEEE Std 802.11h-2004, 2004.

[75] IEEE. Air Interface for Fixed and Mobile Broadband Wireless Access Systems. IEEE P802.16e/D12, February 2005.

[76] IEEE. IEEE 802.21 Media Independent Handover Services, IEEE P802.21/D01.00, March 2006.

[77] IEEE. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. High Throughput. Draft 3.2 IEEE 802.11n, 2009.

[78] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafisov. TCP over Second (2.5G) and Third (3G) Generation Wireless Networks. Internet RFCs, ISSN 2070-1721, RFC 3481, February 2003.

[79] International Organization for Standardization. Open Systems Interconnection-Basic Reference Model. ISO Standard 7498, 2005.

[80] International Telecommunication Union. NGN Focus Group Proceedings Part I, 2005.

[81] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988.

[82] V. Jacobson. Modified TCP Congestion Avoidance Algorithm. Email to the end2end-interest Mailing List, April 1990. Available at ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[83] I. Järvinen, L. Daniel, and M. Kojo. Experimental Evaluation of Vertical Handoff Algorithms, Unpublished Report. University of Helsinki, Department of Computer Science, April 2008.

[84] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. Internet RFCs, ISSN 2070-1721, RFC 3775, June 2004.

[85] V. Kawadia and P. R. Kumar. A Cautionary Perspective on Cross-layer Design. *IEEE Wireless Communications*, 12(1):3–11, February 2006.

[86] S. Keshav. Why Cell Phones Will Dominate the Future Internet. *ACM Computer Communication Review*, 35(2):83–86, April 2005.

[87] S-E. Kim and J. A. Copeland. TCP for Seamless Vertical Handoff in Hybrid Mobile Data Networks. In *Proceedings of IEEE Globecom 2003*, December 2003.

[88] S-E. Kim and J.A. Copeland. Enhancing TCP Performance for Intersystem Handoff within Heterogeneous Mobile Networks. In *Proceedings of IEEE Vehicular Technology Conference*, May 2004.

[89] L. Kleinrock. Nomadic Computing, Keynote Address. In *International Conference on Mobile Computing and Networking (MobiCom'95)*, 1995.

[90] M. Kojo, K. Raatikainen, M. Liljeberg, J. Kiiskinen, and T. Alanko. An Efficient Transport Service for Slow Wireless Telephone Links.

IEEE Journal on Selected Areas In Communications, 15(7):1337–1348, September 1997.

[91] R. Koodli (ed.). Fast Handovers for Mobile IPv6. Internet RFCs, ISSN 2070-1721, RFC 4068, July 2005.

[92] J. Korhonen, A. Mäkelä, S. Park, and H. Tschofenig. Link and Path Characteristic Information Delivery Analysis. Internet-Draft, October 2006. Work in progress.

[93] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating The Edge Network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC'10)*, 2010.

[94] S. S. Lam. Back to the Future Part 4: Internet. *ACM SIGCOMM Computer Communication Review*, 35(1):3–12, January 2005.

[95] Y. Lin and H. Chang. VA-TCP: A Vertical Handoff-Aware TCP. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 237–238. ACM, 2007.

[96] R. Ludwig and A. Gurtov. The Eifel Response Algorithm for TCP. Internet RFCs, ISSN 2070-1721, RFC 4015, February 2005.

[97] R. Ludwig and R. H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communication Review*, 30(1), January 2000.

[98] R. Ludwig and M. Meyer. The Eifel Detection Algorithm for TCP. Internet RFCs, ISSN 2070-1721, RFC 3522, April 2003.

[99] M. Dillenger and S. Buljor. *Reconfigurable Systems in a Heterogeneous Environment, Software Defined Radio: Architecture, Systems and Functions*. Wiley, 2003.

[100] J. Manner and M. Kojo (Eds.). Mobility Related Terminology. Internet RFCs, ISSN 2070-1721, RFC 3753, June 2004.

[101] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. Internet RFCs, ISSN 2070-1721, RFC 2018, October 1996.

[102] Y. Matsushita, T. Matsuda, and M. Yamamoto. TCP Congestion Control with ACK-Pacing for Vertical Handover. *IEICE Transactions on Communications*, E90-B(4):885–893, 2007.

[103] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. *ACM Computer Communication Review*, 35(2):37–52, April 2005.

[104] D. Molkdar, W. Featherstone, and S. Lambotharan. An Overview of EGPRS: the Packet Data Component of EDGE. *Electronics and Communication Engineering Journal*, 14:21–38, February 2002.

[105] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. Internet RFCs, ISSN 2070-1721, RFC 4423, May 2006.

[106] Network Simulator ns-2. http://www.isi.edu/nsnam/ns.

[107] P. Nikander, T. Henderson, C. Vogt, and A. Jarkko. End-Host Mobility and Multihoming with the Host Identity Protocol. Internet RFCs, ISSN 2070-1721, RFC 5206, 2008.

[108] Nokia. Nokia N8 Specifications. http://europe.nokia.com/find-products/devices/nokia-n8/specifications.

[109] J. Padhye and S. Floyd. On Inferring TCP Behaviour. In *ACM SIGCOMM '01*, August 2001.

[110] K. Pahlavan and P. Krishnamurthy. *Principles of Wireless Networks A Unified Approach*. Prentice-Hall, first edition, 2002.

[111] K. Pahlavan, P. Krishnamurthy, A. Hatami, M. Ylianttila, J. Mäkelä, R. Pichna, and J. Vallström. Handoff in Hybrid Mobile Data Networks. *IEEE Personal Communication*, 7(2):34–47, 2000.

[112] V. Paxson. End-to-end Routing Behavior in the Internet. *SIGCOMM Computer Communication Review*, 36(5):41–56, 2006.

[113] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. Internet RFCs, ISSN 2070-1721, RFC 2988, November 2000.

[114] V. Paxson and S. Floyd. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, December 1997.

[115] C. Perkins. Mobile IP. *IEEE Communications Magazine*, April 2002.

[116] C. Perkins (ed.). IP Mobility Support for IPv4. Internet RFCs, ISSN 2070-1721, RFC 3344, August 2002.

[117] G.P. Pollini. Trends in Handover Design. *IEEE Communications Magazine*, pages 82–90, March 1996.

[118] J. Postel. Transmission Control Protocol. Internet RFCs, ISSN 2070-1721, RFC 793, September 1981.

[119] R. Koodli and C. Perkins. *Mobile Inter-networking with IPv6: Concepts, Principles, and Practices*. Wiley-Interscience, 2007.

[120] R. Moskowitz and P. Nikander and P. Jokela and T. Henderson. Host Identity Protocol. Internet RFCs, ISSN 2070-1721, RFC 5201, 2008.

[121] K. Raatikainen. Wireless Internet: Challenges and Solutions. Technical Report B-2004-03, University of Helsinki, Department of Computer Science, September 2004.

[122] M. Rahnema. Overview of the GSM System and Protocol Architecture. *IEEE Communications Magazine*, 31:92–100, April 1993.

[123] Rysavy Research. HSPA to LTE-Advanced: 3GPP Broadband Evolution to IMT-Advanced (4G). http://www.rysavy.com/Articles/2009_09_3G_Americas_RysavyResearch_HSPA-LTE_Advanced.pdf, September 2009.

[124] H. Rutagemwa, S. Pack, X. Shen, and J. W. Mark. Robust Cross-Layer Design of Wireless-Profiled TCP Mobile Receiver for Vertical Handover. *IEEE Transactions on Vehicular Technology*, 56(6):3899–3911, November 2007.

[125] H. Rutagemwa, M. Shi, X. Shen, and J. W. Mark. Wireless Profiled TCP performance over Integrated Wireless LANs and Cellular Networks. *IEEE Transactions on Wireless Communications*, 6(6):2294–2304, June 2007.

[126] P. Ruuska, J. Mäkelä, M. Jurvansuu, J. Huusko, and P. Mannersalo. ROADMAP for Communication Technologies, Services and Business Models 2010, 2015 and Beyond. http://www.tekes.fi/u/GIGA-Roadmap2010.pdf, August 2010.

[127] A. K. Salkintzis, C. Fors, and R. Pazhyannur. WLAN-GPRS Integration for Next-generation Mobile Data Networks. *IEEE Wireless Communications Magazine*, 9(5):112–124, October 2002.

[128] S. Saroiu, P.K. Gummadi, and S.D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking*, pages 156–170, January 2002.

[129] P. Sarolahti, M. Allman, and S. Floyd. Determining an Appropriate Sending Rate Over an Underutilized Network Path. *Computer Networks (Elsevier)*, May 2007.

[130] P. Sarolahti, S. Floyd, and M. Kojo. Transport-layer Considerations for Explicit Cross-layer Indications. Internet-Draft, March 2007. Work in progress.

[131] P. Sarolahti and M. Kojo. Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and Stream Control Protocol (SCTP). Internet RFCs, ISSN 2070-1721, RFC 5682, September 2009.

[132] P. Sarolahti, M. Kojo, and K. Raatikainen. F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts. *ACM SIGCOMM Computer Communication Review*, 33(2):51–63, April 2003.

[133] P. Sarolahti, J. Korhonen, L. Daniel, and M. Kojo. Using Quick-Start to Improve TCP Performance with Vertical Hand-offs. In *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, pages 897–904, November 2006.

[134] P. Sarolahti and A. Kuznetsov. Congestion Control in Linux TCP. In *Proceedings of Usenix 2002/Freenix Track*, pages 49–62, Monterey, CA, USA, June 2002.

[135] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *Computer Communication Review*, 29(5):71–78, October 1999.

[136] H. Schulzrinne and E. Wedlund. Application-Layer Mobility Using SIP. *ACM Mobile Computing and Communications Review*, July 2000.

[137] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol Enhancements for Intermittently Connected Hosts. *ACM Computer Communication Review*, 35(2), July 2005.

[138] S. Schütz, N. Koutsianas, L. Eggert, W. Eddy, Y. Swami, and K. Le. TCP Response to Lower-Layer Connectivity-Change Indications. Internet-Draft, February 2008. Work In Progress.

[139] E. Seurre, P. Savelli, and P.-J. Pietri. *EDGE for Mobile Internet*. Artech House, 2003.

[140] S. Sharma, I. Baek, and T. Chiueh. OmniCon: A Mobile IP-based Vertical Handoff System for Wireless LAN and GPRS Links. *Software-Practice and Experience*, 37:779–798, June 2007.

[141] F. Siddiqui and S. Zeadally. Mobility Management Across Hybrid Wireless Networks: Trends and Challenges. *Computer Communications*, 29(9):1363–1385, 2006.

[142] A. Snoeren and H. Balakrishnan. An End-to-End Approach to Host Mobility. In *Proceedings of the sixth ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 155–166, August 2000.

[143] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhauser, 2004.

[144] K. Sripanidkulchai, B. Maggs, and H. Zhang. An Analysis of Live Streaming Workloads on the Internet. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC'04)*, pages 41–54, 2004.

[145] M. Stemm and R. H. Katz. Vertical Handoffs in Wireless Overlay Networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

[146] W. Stevens. *TCP/IP Illustrated, Volume 1; The Protocols*. Addison Wesley, 1995.

[147] R. Stewart, et. al. Stream Control Transmission Protocol. Internet RFCs, ISSN 2070-1721, RFC 2960, October 2000.

[148] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10–23, November/December 1997.

[149] K. Tsukamoto, Y. Fukuda, Y. Hori, and Y. Oie. New TCP Congestion Control Scheme for Multimodal Mobile Hosts. *IEICE Transactions on Communications*, E89-B(6):1825–1836, 2006.

[150] M. Weiser. The Computer for the Twenty-first Century. *Scientific American*, pages 94–104, September 1991.

[151] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, July 1993.

[152] WiMAX. Worldwide Interoperability for Microwave Access (WiMAX). www.wimaxforum.org.

[153] WiMAX. Mobile WiMAX - Part I: A Comparative Analysis. WiMax Forum, May 2006.

[154] WiMAX. Mobile WiMAX - Part I: A Technical Overview and Performance Evaluation. WiMax Forum, August 2006.

[155] G. Wright and W. Stevens. *TCP/IP Illustrated, Volume 2; The Implementation*. Addison Wesley, 1995.

[156] X. Wu, M-C. Chan, and A. L. Ananda. Improving TCP Performance in Heterogeneous Mobile Environments by Exploiting the Explicit Cooperation Between Server and Mobile host. *Computer Networks*, 52:3062–3074, 2008.

[157] M. Ylianttila, M. Pande, J. Mäkelä, and P. Mähönen. Optimizaion Scheme for Mobile Users Performing Vertical Handoffs between IEEE 802.11 and GPRS/EDGE Networks. In *Proc. IEEE Globecom 2001*, volume 6, pages 3439–3443. IEEE, December 2001.

[158] M. Zhang, W. John, K. Claffy, and N. Brownlee. State of the Art in Traffic Classification: A Research Review. In *Proceedings of the PAM Student Workshop*, 2009.

[159] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. In *Proc. of the Eleventh IEEE International Conference on Networking Protocols (ICNP 2003)*, 2003.

I

Paper 1

Laila Daniel and Markku Kojo

Adapting TCP for Vertical Handoffs in Wireless Networks

In Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06), November 2006, pages 151 - 158.

Adapting TCP for Vertical Handoffs in Wireless Networks

Laila Daniel, Markku Kojo
Department of Computer Science
P.O. box 68, 00014
University of Helsinki, Finland
{laila.daniel,markku.kojo}@cs.helsinki.fi

Abstract

With the growing use of multi-radio mobile terminals a vertical handoff between different wireless access technologies is becoming increasingly common. The vertical handoff may result in a significant change in the access link characteristics that can affect the performance of TCP dramatically as its behaviour depends on the end-to-end path properties that also change consequently. We propose a number of simple enhancements to the TCP sender algorithm which make use of explicit information about the change in the link characteristics due to handoff. We study the effectiveness of the enhancements in a simulated WLAN-GPRS environment with different handoff scenarios. The enhancements are shown to improve TCP performance significantly.

1 Introduction

Internet access using mobile devices is increasingly popular with the introduction of multi-radio mobile nodes (MNs) equipped with multiple interfaces to access networks using diverse link technologies. For example, an MN with interfaces to Wireless LAN (WLAN) and Wireless WAN (WWAN) can select the appropriate access network based on the connectivity or performance requirement of the application. Switching between access networks is referred to as handoff and it can be categorized as horizontal handoff and vertical handoff. Horizontal handoff involves switching within the same access technology whereas vertical handoff involves switching between different access technologies [10].

Access networks with different link layer technologies vary widely in their characteristics such as link bandwidth, latency, bit-error rate and the degree of bandwidth asymmetry. The wireless access link is commonly the last-hop (first-hop) link and is usually the bottleneck link on the end-to-end path. A significant change in the access link characteristics easily affects the end-to-end path properties and thereby the behaviour of transport protocols. In the case of Transmission Control Protocol (TCP) [16], the most widely used transport protocol in the Internet, vertical handoff may incur packet losses, intermittent connectivity, packet reordering and spurious or too late retransmission timeouts (RTOs), resulting in unnecessary TCP congestion response or inefficient loss recovery that sacrifice TCP performance [4, 6, 7, 8, 18].

When a vertical handoff occurs, the TCP sender adjusts its transmission rate and RTO estimate very slowly to the new end-to-end path as it learns the properties of the new path implicitly by probing it over several round trips. If the TCP layer is explicitly notified about the changes in the path properties, the TCP sender could react more timely and efficiently and possibly avoid false congestion responses. The TCP layer on the MN can be locally notified of the changes in the attached access link characteristics, but the TCP layer on the corresponding node (CN) is completely unaware of the changes. In order to inform the TCP sender on the CN some proposals have been introduced recently for delivering information from the MN to the CN about the path change due to a handoff [11, 14, 18, 20]. While learning the new access link characteristics cannot be used to reliably determine the new end-to-end path properties, learning about significant changes in the last-hop (first-hop) link characteristics can be used as a useful hint about a potential change in the end-to-end path properties.

In this paper we study the problems of TCP due to a vertical handoff in a WLAN-GPRS/EDGE(EGPRS) [3, 19] environment. We propose a set of TCP sender algorithms that allow the TCP sender to cope with the problems arising due to the vertical handoff. These algorithms are invoked as a response to an explicit indication notifying the TCP sender about significant change in the access link bandwidth and/or delay. It is pertinent to point out that the information about the change in the link characteristics need not be precise but it is sufficient to indicate changes of an order of magnitude. In addition, we take a conservative approach with our algorithms by making the TCP sender to react more

1-4244-0419-3/06/$20.00 ©2006 IEEE

conservatively than the regular TCP reacts in most cases and by being careful when making TCP more aggressive. The effectiveness of the algorithm is studied in a simulated WLAN-EGPRS environment. We apply the proposed algorithms in different handoff scenarios with change in delay and bandwidth of the last-hop link and analyze TCP performance immediately after the handoff. Our results show that with the proposed algorithms we can achieve 20-60% reduction in transfer time.

The rest of the paper is organized as follows. Section 2 gives an overview of the related work. Section 3 discusses the various aspects of the proposed algorithms for TCP sender. Section 4 evaluates the proposed algorithms in different handoff scenarios between EGPRS and WLAN access networks. Section 5 presents our conclusions.

2 Related Work

A vertical handoff can be of two types based on the connectivity to the old access router during the handoff, namely, break-before-make and make-before-break [10]. In the break-before-make handoff, the connection to the old access router breaks before the handoff completes, causing disruption in connectivity and often resulting in packet losses. In the worst case, the entire window of TCP segments is lost. By contrast in the make-before-break handoff, only after the connection to the new access router is operable, the connection to the old access router may be torn down. Hence, no packet losses occur.

A practical study on the performance of TCP with vertical handoff between GPRS and WLAN is presented in [4]. It points out that the period of disconnection in a break-before-make handoff often causes TCP to timeout and contributes to the degradation in TCP performance. The high buffering in GPRS aggravates the performance degradation as it inflates the round-trip time (RTT) and RTO values.

A thorough study on the effect of the make-before-break handoff on TCP performance is given in [7]. The paper suggests a *nodupack* scheme as a solution to the problem of packet reordering which occurs with a handoff from a high-delay to a low-delay link. This scheme suppresses the transmission of duplicate acknowledgements (dupacks) as a response to the out-of-order packets arriving through the new low-delay link during the handoff. Reducing the congestion window (*cwnd*) is proposed as a solution to overcome the packet losses due to the bandwidth-delay product (BDP) decrease.

A comparative study of the effect of vertical handoff on transport protocols is presented in [6]. It shows that transport protocols have difficulties in adapting to the network after handoff because of the decrease in the link BDP. Vertical handoff is modelled by changing the bandwidth and delay of the link and changing the access router buffer size to reflect the size of the new access router buffer. This type of modeling cannot simulate the break-before-make handoff and does not simulate the make-before-break handoff correctly in case the new buffer is smaller than the old buffer as the excess packets in the buffer are discarded if the new buffer cannot hold all queued packets. To reduce the problem due to the differences in the link BDPs the paper proposes overbuffering all nodes along the path using the maximum link BDP on the end-to-end path. A drawback of this scheme is that the knowledge of all link BDPs is not available for all nodes on the path. In addition, overbuffering is generally not advisable as it increases queuing delay.

The paper [8] identifies packet reordering, BDP mismatch and spurious RTOs as the problems arising from a make-before-break handoff. The paper proposes three schemes, *fast response*, *slow response*, and *ack delaying* to solve the problem of spurious RTOs due to the handoff from a low-delay link to a high-delay link. The basic idea behind these schemes is to reduce the difference in RTT between the old and new link. In the fast response scheme, after the handoff, the old low-delay link is used for a short period for sending the ACKs for the first few packets arriving over the new high-delay link. In the slow response scheme, the RTTs for the last few packets that arrive through the old low-delay link is increased by starting to send the ACKs over the new high-delay link already before the handoff is completed. The problem with these schemes is that the period for which the ACKs are sent through the old/new link is to be determined carefully so that the RTT estimate will have a high enough value to avoid the spurious RTOs. Nonetheless, if the difference in delay is large enough these schemes cannot avoid spurious RTOs. In the ACK delaying scheme, the ACKs of the few last low-delay link packets are delayed at the IP layer. However, a proper value for the delay period is difficult to determine correctly as the new end-to-end RTT is unknown.

The Internet draft [14] introduces a model for delivering link characteristics information from an MN to CNs during the handoff procedure. In Mobile IPv6 this information is included in the Binding Update message and sent to the CN when a vertical handoff occurs and/or when there is a significant change in the link characteristics.

The lightweight mobility detection and response (LMDR) algorithm [20] proposes to make TCP aware of the path change due to a vertical handoff. It is assumed that the MN notices the path changes either from lower layers or through other out-of-band mechanisms. This information is relayed to the TCP sender on the CN through a new TCP option. After the path change, the TCP sender resets the congestion control state, RTT variables and RTO timer as specified for a new connection in RFC2988 [15], sends an initial window worth of data over the new path, and continues in slow-start.

The paper [18] presents an experimental study in a mobile environment where the MN moves often and intermittent connectivity due to handoffs is common. The authors observe that the major problem with the intermittent connectivity is the TCP timeouts and retransmission behavior that can significantly decrease TCP performance or even cause connections to abort. The paper proposes Host Identity Protocol (HIP) [13] as a solution to host mobility. If the disconnection period during a handoff is longer than the *user timeout* of TCP, the TCP sender will abort the connection. They propose a *TCP user timeout* option with which the MN can specify a longer user timeout value to avoid a connection abort during a prolonged disconnected period. Another proposal, *TCP retransmission trigger* is used to trigger the TCP sender to retransmit immediately after the disconnected period ends. Otherwise, the TCP sender may wait for a long time until the backed-off retransmission timer expires, resulting in a long unused connection time. The trigger can be implemented by sending three gratuitous dupacks, by introducing a new *TCP Immediate Retransmission* option, or by using HIP layer to trigger TCP when the HIP readdressing exchange completes.

3 A Discussion of the Proposed Algorithms

The proposals [14, 18, 20] provide the basis of our work to improve TCP performance using the information about the changes in link characteristics due to a vertical handoff. We assume that the TCP sender gets a notification regarding the handoff from the IP layer. This allows timely delivery of the notification to the TCP sender at the CN, as during the handoff the outbound packets from the CN take the new path immediately after the IP mobility registration message, for example the Binding Update message in Mobile IPv6, arrives at the CN. The link characteristics information can be included in the mobility messages [14]. The notification includes information about a significant increase or decrease in the access link bandwidth and/or delay. We categorize the problems of TCP due to vertical handoff into 3 classes, namely, problems due to changes in delay, due to changes in bandwidth and due to intermittent connectivity and propose enhancements to the TCP sender algorithm to alleviate these problems.

3.1 Handoff and Changes in Delay

Handoff from a low delay to high-delay link

When a make-before-break handoff occurs from a low-delay link to a high-delay link, spurious RTOs occur due to the significant change in RTT [8]. During the make-before-break handoff, there are no packet losses but TCP sender times out spuriously and retransmits a large number of packets unnecessarily. The small RTO value of the low-delay path causes timeout to occur as the ACKs for the packets sent before the handoff take the high-delay link after the handoff. The spurious RTO causes *ssthresh* to become half of *FlightSize* and *cwnd* to one, resulting in performance degradation.

```
When handoff notification arrives indicating
 significant increase in delay:
    If (TCP is not in RTO recovery)
       Maintain RTO value at 3 seconds until all
       segments sent before handoff have been ACKed
       Initialize RTT variables as for a new connection
      When ACK for a new data segment arrives
        Update RTT variables
```

Figure 1. incRTO algorithm

The *incRTO* algorithm given in Figure 1 aims at avoiding spurious RTOs. This algorithm is invoked when the handoff notification indicating a significant increase in delay arrives at the TCP sender and the TCP sender is not in RTO recovery at that moment. The algorithm maintains the RTO value at 3 seconds until the ACKs for all the segments sent before the handoff notification have been received. At that point, the RTT variables are initialized as specified for a new connection [15]. The choice of the RTO value of 3 seconds is based on RFC 2988 which recommends an RTO value of 3 seconds until a proper RTT is obtained. When ACK for a segment sent through the new link arrives, the RTT variables are immediately updated. The case of TCP being in RTO recovery when the handoff notification arrives is treated as a case of intermittent connectivity and is discussed in Section 3.3.

Handoff from a high-delay to a low-delay link

Packet reordering is a problem that TCP faces when a make-before-break handoff takes place from a high-delay link to a low-delay link [7]. The packets sent through the low-delay link after handoff may overtake the packets transmitted through the high-delay link before handoff and this packet reordering generates dupacks. If the TCP sender receives a *dupThresh* number of dupacks (typically 3) it retransmits the first unacknowledged packet. TCP halves the *ssthresh* and *cwnd* and continues in fast recovery until all the packets sent before handoff are ACKed. The retransmission and the reduction in *ssthresh* and *cwnd* are unnecessary as the dupacks which arrive are due to packet reordering and not due to congestion losses. DSACK [2] and Eifel [12] can be thought as solutions to this problem as they undo the unnecessary congestion control actions due to packet reordering. A proactive action called nodupack scheme which avoids sending the dupacks and thereby avoiding the unnecessary congestion control measures is given in [7].

A more serious consequence of this handoff scenario

```
When handoff notification arrives indicating
 significant decrease in delay:
    If(TCP is not in RTO recovery)
      Wait till the segments sent before
       handoff have been ACKed
       Initialize RTT variables as for a new connection
      When ACK for a new data segment arrives
        Update RTT variables
```

Figure 2. RTO-conv algorithm

may be the inflated RTO value. The high-delay link already has a high RTT and the high buffering (if present) increases the RTO value [4]. If the handoff from a high-delay link to a low-delay link occurs in the slow-start phase or near the buffer overflow points in the congestion avoidance phase, RTO may be very high due to buffering. After the handoff, the RTO will converge to the low RTO of the low-delay path very slowly, as the RTT variables are updated once in an RTT [15]. If the TCP sender is in slow-start when a handoff to lower BDP link occurs, a slow-start overshoot [5] may cause very many packets to be dropped and an RTO recovery may be needed to recover the lost packets. An RTO is needed also if a retransmitted segment becomes dropped, e.g., due to a link error. Invoking the RTO recovery will take a longer time due to the high RTO value. The algorithm *RTO-conv* given in Figure 2 is intended to overcome this problem. When all the packets sent before handoff have been ACKed, we initialize the RTT variables as recommended for a new link. and we update the RTT variables immediately on a new ACK. This speeds up the convergence to the low RTO value of the low-delay path.

3.2 Handoff and Changes in Bandwidth

Handoff from a high bandwidth to a low bandwidth link

When a handoff occurs from a high-BDP to a low-BDP link, a TCP sender unaware of this change, injects more packets to the network than the capacity of the low-BDP link and many packets may get dropped [6, 7]. When the handoff occurs from a high-bandwidth link to a low-bandwidth link, the new link potentially has lower BDP and becomes congested, resulting in many packets drops. To avoid this problem we introduce the *cwnd-reduction* algorithm (see Figure 3) that makes TCP less aggressive by reducing the *cwnd* by half. As a result the TCP sender will inject fewer packets to the low-bandwidth link which helps to reduce packet losses.

```
When handoff notification arrives indicating
a significant decrease in bandwidth:
    If( TCP not in fast recovery or in RTO recovery)
      cwnd = max(2, cwnd/2);
```

Figure 3. cwnd-reduction algorithm

This *cwnd* reduction takes place only if TCP is not in fast recovery or RTO recovery when the handoff notification arrives as TCP has already reduced *cwnd* in these cases. The link BDP may not always change although there is a considerable change in bandwidth. For example, if delay changes to the opposite direction at the same time, the BDP may remain roughly the same. Here we take a conservative approach and reduce the *cwnd*.

Handoff from a low bandwidth to a high bandwidth link

In this scenario, TCP's inability to efficiently utilize the high bandwidth available is the main problem. An attempt to make TCP more aggressive by increasing *cwnd* is not a viable approach without proper view of the end-to-end path conditions. One possibility is to employ an enhanced version of TCP Quick-Start as a solution to this problem [17].

3.3 Handoff and Intermittent Connectivity

When a break-before-make handoff occurs, the MN has no connection to any access router. As a result the end-to-end path between the MN and CN is broken and the connection is up again only after the handoff completes. During this disconnection period, the TCP sender will not get ACKs for the packets it has already transmitted. If the TCP sender has unacknowledged data at the expiry of the retransmission timer, it retransmits the first unacknowledged packet and doubles the RTO value. TCP doubles the RTO value after each retransmission attempt [15]. When the end-to-end connection is up again, TCP sender waits until the RTO expires before attempting another retransmission. The unused connection time can be up to one minute depending on the disconnection length and the next scheduled RTO. This unused connection time increases the recovery time of the lost packets.

We propose an algorithm called *rxmt-immediate* given in Figure 4 to avoid this problem. This algorithm retransmits the first unacknowledged packet immediately if the

```
On the first expiration of RTO:
  Save ssthresh

When handoff notification arrives:
  If (TCP in RTO recovery)
    Retransmit the first unacknowledged packet
    Restore ssthresh
    If there is a significant change in delay
      Initialize RTT variables as for a new connection
      When ACK for new data arrives
        Update RTT variables
```

Figure 4. rxmt-immediate algorithm

TCP sender is in RTO recovery when the handoff notification arrives. According to the TCP congestion control algorithms [1], after a timeout the TCP sender will set *ssthresh* to *FlightSize*/2 and *cwnd* to 1. If more than one timeout occurs, *ssthresh* value is further reduced in many implementations. The purpose of this reduction in *ssthresh* and *cwnd* is to account for the severity of the congestion. But in the break-before-make handoff, the RTO occurs more than once due to disconnection and not due to congestion. Therefore, the *ssthresh* is restored using the *ssthresh* value after the first expiration of RTO in the *rxmt-immediate* algorithm, that is, the *ssthresh* becomes reduced only on the first RTO. In addition, if there is a significant change in delay we initialize the RTT variables as for a new connection. The RTT variables are updated immediately when the first ACK for a data segment sent after the handoff arrives.

4 Evaluation of the Proposed Algorithms in WLAN-EGPRS Environment

We examine the performance of the proposed algorithms in a simulated WLAN-EGPRS environment using ns-2 [9] network simulator. The WLAN bandwidth ranges between 1 - 50 Mbps though the higher bandwidths are not realized in practice and one way propagation delay ranges roughly between 1 - 10 ms. The EGPRS bandwidth ranges from 128 to 473 Kbps (theoretical maximum) and one way propagation delay ranges from 300 ms to 350 ms.

4.1 Simulation Setup

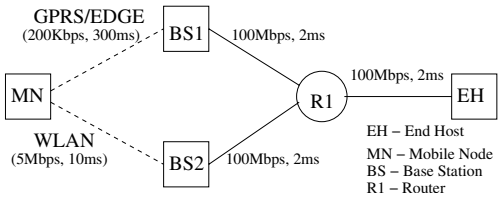


Figure 5. Network Topology for Handoff Tests

In this study, we take the WLAN bandwidth to be 5 Mbps and one way propagation delay of 10 ms whereas the corresponding parameters for EGPRS are chosen to be 200 Kbps and 300 ms. For this combination of bandwidth and delay, we can see that the BDPs of both links are nearly the same though the ratios of change in bandwidth and delay with EGPRS-WLAN handoff are 1:25 and 30:1, respectively. This corresponds to an order of magnitude change and is seen as 'a significant change' in our algorithms.

The network topology used in our experiments is shown in Figure 5. MN can be attached to both WLAN and EGPRS wireless access technologies. Both EGPRS and WLAN links have dedicated base stations, BS1 and BS2, which are connected to a common wireless access router R1 via 100 Mbps links. The router has a 100 Mbps link to CN. The one way propagation delay of the fixed links is 2 ms. The router buffer for WLAN and EGPRS are 30 and 32 packets, respectively. As ns-2 supports routing updates on the fly by changing the route metrics for the links, we model the handoff by changing the route the packets take before and after the handoff.

We consider a single TCP flow from CN to MN. The TCP packet size is 1500 bytes which includes the headers. In our simulation we consider handoff to occur in all phases of a TCP connection, namely, at the beginning of a connection, just before and during the slow-start overshoot, and in the congestion avoidance phase. For WLAN to EGPRS handoff, the handoff occurs once during the first 10 seconds of the TCP connection at any of the 100 observation points at 10 ms intervals whereas the EGPRS to WLAN handoff occurs during the first 20 seconds to allow all phases of a TCP connection also with the slower EGPRS. Both make-before-break as well as break-before-make handoffs are examined. For break-before-make handoff the period of disconnection is taken to be 500 ms.

We compare TCP performance with and without the proposed enhancements by measuring the elapsed time to transfer 'n' packets after handoff where n varies from 50 to 200. We report the results only for the case where n is 100 as the results are very similar in all cases. As the baseline TCP, we use the standard TCP Sack1 algorithm given in ns-2 simulator.

4.2 Handoff from WLAN to EGPRS

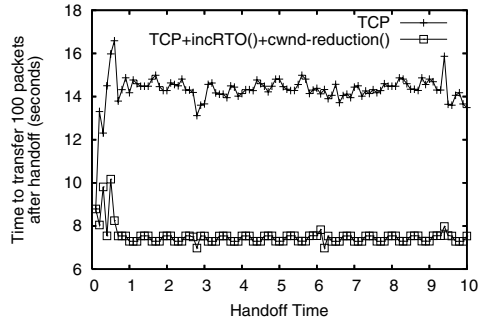
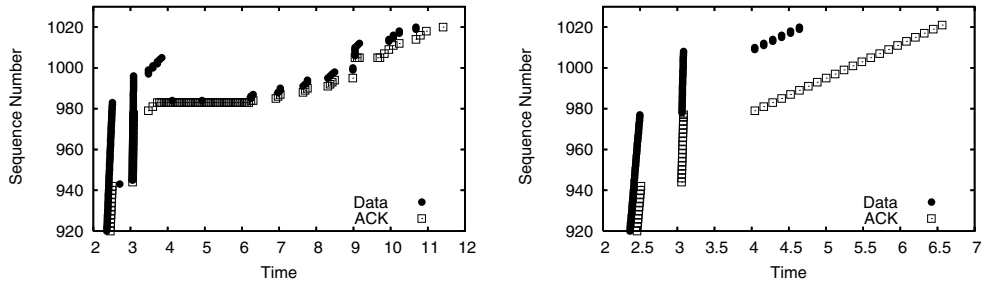


Figure 6. Time taken to transfer 100 packets after a make-before-break handoff from WLAN to EGPRS

Figure 6 summarizes the results by providing the time taken for transferring 100 packets after a make-before-break



(a) TCP (b) TCP+incRTO+cwnd-reduction

Figure 7. WLAN - EGPRS make-before-break handoff - The problem of Spurious RTO is shown in (a) with regular TCP. This problem is eliminated in (b) with incRTO+cwnd-reduction

handoff. The x-axis shows the time at which handoff occurs after the beginning of the connection. The y-axis shows the elapsed time in seconds to transfer 100 packets after the handoff. In this test the algorithms *incRTO* and *cwnd-reduction* become effective.

When the handoff occurs during the congestion avoidance phase from 0.7 second onwards the reduction in transfer time with our algorithms is due to avoiding the harmful effect of the spurious RTOs and the reduction in packet losses due to *cwnd* reduction, yielding approximately 45% reduction in transfer time.

During the slow-start phase (0 to 0.7 second), especially around the slow-start overshoot, both regular TCP and TCP with our algorithms are required to recover a large number of packets that are dropped due to slow-start overshoot, resulting in inefficient progress after the handoff. However, the regular TCP cannot avoid the spurious RTO that occurs either just before the slow-start overshoot or during the recovery from the slow-start overshoot. In the former case, the regular TCP behaves over-aggressively as the spurious RTO results in unnecessary retransmission of several packets at a high rate, making the overshoot even worse. In the latter case, the handoff occurs during the SACK-based fast recovery and with the low RTO value the retransmission timer expires several times spuriously during the recovery, forcing the TCP sender to continue the recovery slowly with a small *cwnd* and reduced *ssthresh* value.

Figure 7(a) provides a closer look into a specific instance of WLAN-EGPRS make-before-break handoff. The handoff occurs after 2.5 seconds when the TCP connection is in the congestion avoidance phase. TCP timeouts spuriously at 2.7 seconds. The ACKs of the original transmissions start arriving over the EGPRS link at 3.05 seconds and trigger TCP to retransmits about 37 packets unnecessarily. As the ACKs are arriving continuously, TCP continues by sending new packets overloading the new link and about 20 packets are lost. After this, the dupacks for unnecessary retransmissions start arriving and RTO expires again twice before the RTO recovery starts to make progress. As the RTO expires more than once, in ns-2 the ssthresh value is reduced to 2 forcing the TCP sender to continue with a small *cwnd* for a long time.

TCP with *incRTO* avoids the spurious RTO as the RTO value is set to 3 seconds till the ACKs for all segments sent before the handoff have arrived. From Figure 7(b) we observe that there are no unnecessary retransmissions. The TCP sender continues to send new data when it starts to get the ACKs over the EGPRS link. The *cwnd-reduction* algorithm is also executed here and as the *cwnd* is halved, there are no additional packet losses.

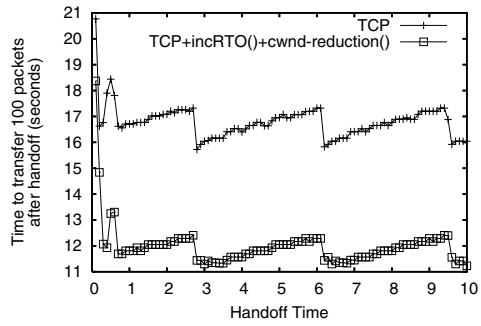
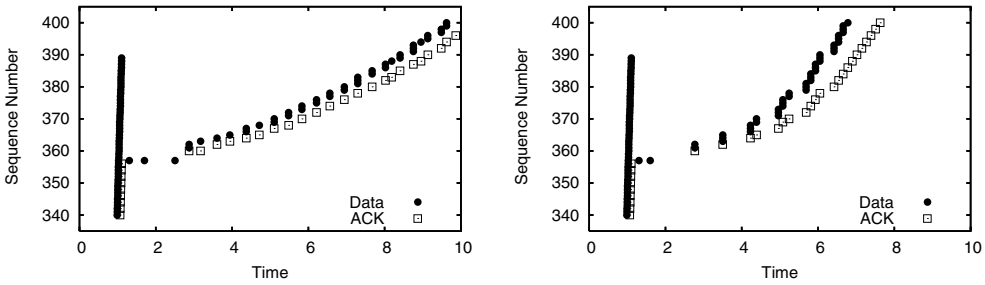


Figure 8. Time taken to transfer 100 packets after a break-before-make handoff from WLAN to GPRS

The results with the break-before-make handoff from WLAN to EGPRS are illustrated in Figure 8. When the handoff occurs in the congestion avoidance phase (starting after 0.7 second), TCP with *rxmt-immediate* is able to reduce the time to transfer 100 packets after the handoff by



(a) TCP (b) TCP+rxmt-immediate

Figure 9. WLAN - EGPRS break-before-make handoff - The problem of Prolonged disconnection is shown in (a). The rxmt-immediate algorithm reduces the handoff delay as shown in (b)

about 30%. This is due to the fact that the RTO expires during the disconnection and the *rxmt-immediate* algorithm immediately retransmits when the handoff notification arrives and reduces the *ssthresh* value only once, decreasing the delay in recovering the lost packets and allowing the TCP sender to continue with larger window. When the handoff occurs during the slow-start, TCP has to recover the packet drops due to slow-start overshoot in addition to the losses due to the disconnection, resulting in worse and unstable results with both TCP variants.

Figure 9(a) shows the time-sequence graph of a break-before-make handoff occurring at 1.1 seconds with the regular TCP. The TCP sender timeouts and retransmits at 1.3, 1.7 and 2.5 seconds. The first retransmission is lost due to disconnection. The second retransmission makes it to the receiver, but the RTO expires the third time before the ACK arrives at the TCP sender. Because of the multiple timeouts the *ssthresh* becomes 2 and *cwnd* 1, and the TCP sender continues the RTO recovery very slowly with small *cwnd*.

The same scenario employing TCP with *rxmt-immediate* is shown in Figure 9(b). The second retransmission occurs at 1.6 seconds triggered by the handoff notification. The *ssthresh* value stored by the *rxmt-immediate* algorithm is 16, allowing the TCP sender to recover much faster in slow start.

4.3 Handoff from EGPRS to WLAN

In the case of make-before-break handoff from EGPRS to WLAN, in the present simulation setup, TCP with our enhancements behaves in a similar way as regular TCP as no RTO recovery is needed and therefore the *RTO-conv* algorithm is not invoked. This is because the SACK-based fast recovery is able to recover all losses due to the slow-start overshoot as well as later losses during the congestion avoidance.

The results for the break-before-make handoff from EGPRS to WLAN with a 500 ms disconnection period showed that TCP with our enhancements behaves similarly to the regular TCP. This is because the RTO value is large in EGPRS, and a timeout will not occur during this short disconnection. Therefore, TCP will not enter the *rxmt-immediate* algorithm.

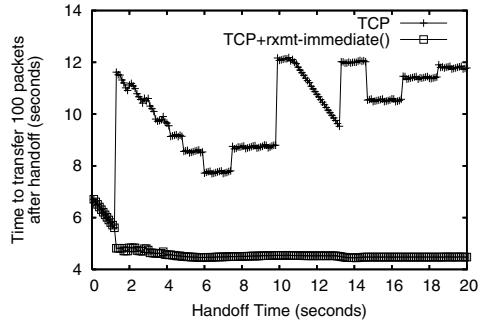


Figure 10. Time taken to transfer 100 packets after a break-before-make handoff from EGPRS to WLAN - Disconnection period 4 seconds

We run the simulations also with longer disconnection periods. Starting from the disconnection period of 2.5 - 3 seconds that is roughly the same as the RTO value in use with EGPRS, the *rxmt-immediate* algorithm starts to become effective. With a disconnection period of 4 seconds and longer, our algorithm becomes consistently effective and significantly improves the TCP performance.

The results with a 4-second disconnection period are shown in Figure 10. In this case, the rxmt-immediate() algorithm improves the performance by about 40-60 % in all

other cases except in the very beginning of the connection as ns-2 initializes the RTO and RTT variables with overly high value and the RTO expires only after the handoff completes.

5 Conclusions and Future Work

In this paper, we have proposed enhancements to the TCP sender algorithm to improve TCP performance in the presence of vertical handoff with the help of notifications about the significant changes in the access link characteristics. We have grouped the changes due to vertical handoff as arising from changes in delay, bandwidth and connectivity and proposed enhancements to adapt TCP to various handoff scenarios. Our solutions effectively addressed the problems arising from spurious RTOs, packet losses, prolonged disconnection and slow convergence to the new RTO value due to a handoff. The proposed enhancements have been evaluated in both make-before-break and break-before-make handoffs in WLAN-EGPRS environment and the simulation results show significant performance gain with the enhancements in the majority of the test cases.

Moving to the higher bandwidth environment is challenging to TCP as TCP needs to be more aggressive but this cannot be done safely based on the access link characteristics only. A TCP sender should probe the new network path, but often it takes relatively long before TCP adapts to the new path. In the future, we intend to study how TCP could use more detailed notifications regarding the changed access link characteristics as useful hints, combine the hints with the information gathered through probing the new network path and thereby try to converge faster but safely to the new end-to-end RTT and available network capacity. In addition, we intend to run experiments in a real network environment with EGPRS and WLAN access networks.

Acknowledgements

We would like to thank Pasi Sarolahti for his help in developing the simulation model. We would also like to thank our colleagues, Lauri Hyttinen, Ilpo Järvinen and Aki Nyrhinen in the IIP Cross project for the fruitful discussions and feedback.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, Apr. 1999.

[2] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM Computer Communication Review*, 32(1), Jan. 2002.

[3] G. Brasche and B. Walke. Concepts, services and protocols of the new GSM phase 2+ general packet radio service. *IEEE Communications Magazine*, 35(8):94–104, Aug. 1997.

[4] R. Chakravorty, P. Vidales, K. Subramanian, I. Pratt, and J. Crowcroft. Performance Issues with Vertical Handovers: Experiences from GPRS Cellular and WLAN hot-spots Integration. In *Proc. IEEE Pervasive Communications and Computing Conference, (IEEE PerCom 2004)*, Mar. 2004.

[5] S. Dawkins, G. Montenegro, M. Kojo, and V. Magret. End-to-end Performance Implications of Slow Links. RFC 3150, July 2001.

[6] A. Gurtov and J. Korhonen. Effect of Vertical Handovers on Performance of TCP-Friendly Rate Control. *ACM Mobile Computing and Communications Review*, 8(3):73–87, July 2004.

[7] W. Hansmann and M. Frank. On Things to Happen During a TCP Handover. In *Proc. 28th IEEE Conference on Local Computer Networks, (LCN'03)*, Oct. 2003.

[8] H. Huang and J. Cai. Improving TCP Performance during Soft Vertical Handoff. In *Proc. 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, volume 2, pages 329–332, Mar. 2005.

[9] ISI at University of South California. Network simulator 2. Available at: http://www.isi.edu/nsnam/ns/.

[10] J.Manner and M. Kojo. Mobility Related Terminology. RFC 3753, June 2004.

[11] J. Korhonen, S. Park, J. Zhang, C. Hwang, and P. Sarolahti. Link Characteristic Information for IP Mobility Problem Statement. Internet-Draft "draft-korhonen-mobopts-link-characteristics-ps-01.txt", June 2006. Work in progress.

[12] R. Ludwig and R. H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communication Review*, 30(1), Jan. 2000.

[13] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol. Internet-Draft "draft-ietf-hip-base-05", Mar. 2006. Work in progress.

[14] S. Park, M. Lee, J. Korhonen, and Z. Zhang. Link Characteristic Information for Mobile IP. Internet-Draft "draft-daniel-mip-link-characteristics-02.txt", June 2005. Work in progress.

[15] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988, Nov. 2000.

[16] J. Postel. Transmission Control Protocol. RFC 793, Sept. 1981.

[17] P. Sarolahti, J. Korhonen, L. Daniel, and M. Kojo. Using Quick-Start to Improve TCP Performance with Vertical Hand-offs. In *Proc. 31st IEEE Conference on Local Computer Networks, (LCN'06)*, Nov. 2006.

[18] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol Enhancements for Intermittently Connected Hosts. *ACM Computer Communication Review*, 35(2), July 2005.

[19] E. Seurre, P. Savelli, and P.-J. Pietri. *EDGE for Mobile Internet*. Artech House, 2003.

[20] Y. Swami, K. Le, and W. Eddy. Lightweight Mobility Detection and Response (LMDR) Algorithm for TCP. Internet-Draft "draft-swami-tcp-lmdr-07.txt", Feb. 2006. Work in progress.

Paper 2

II

Laila Daniel and Markku Kojo

Employing Cross-layer Assisted TCP Algorithms to Improve TCP Performance with Vertical Handoffs

In International Journal on Communication Networks and Distributed Systems (IJCNDS), Vol. 1, No.4/5/6, November 2008, pages 433-465

Employing cross-layer assisted TCP algorithms to improve TCP performance with vertical handoffs

Laila Daniel* and Markku Kojo

Department of Computer Science,
University of Helsinki,
P.O. Box 68, 00014, Finland
E-mail: laila.daniel@cs.helsinki.fi
E-mail: markku.kojo@cs.helsinki.fi
*Corresponding author

Abstract: In this paper we study the performance of TCP with a vertical handoff between access networks with widely varying link characteristics. TCP being an end-to-end protocol has performance problems as its behaviour depends on the end-to-end path properties which are likely to be affected by a vertical handoff. We propose a set of enhancements to the TCP sender algorithm that leverage on explicit cross-layer information regarding the changes in the access link delay and bandwidth. We carry out a systematic study to identify the problems of regular TCP and compare the performance of the regular TCP with the performance of the enhanced TCP algorithms in various handoff scenarios between access networks having different bandwidth and delay characteristics. Our experiments show that with cross-layer notifications TCP performance can be improved significantly in most vertical handoff scenarios.

Keywords: vertical handoff; TCP; wireless networks; cross-layer notifications; performance analysis.

Reference to this paper should be made as follows: Daniel, L. and Kojo, M. (2008) 'Employing cross-layer assisted TCP algorithms to improve TCP performance with vertical handoffs', *Int. J. Communication Networks and Distributed Systems*, Vol. 1, Nos. 4/5/6, pp.433–465.

Biographical notes: Laila Daniel is a Researcher in the Department of Computer Science, University of Helsinki, Finland and is doing her doctoral studies in the area of mobile and wireless networks. Her research interests are in wireless and mobile networking, network protocols, modelling and simulation of protocols and performance analysis of computer networks.

Markku Kojo is a Senior Research Scientist and Lecturer at the Department of Computer Science in University of Helsinki. In 1994–1995 he worked as a Researcher in the national research project Mowgli being the principal designer of the Mowgli mobile computing architecture that is one of the first in kind in the area of mobile computing. In 1997 he worked for Sonera Ltd. as a Network System Consultant. From 1999 he has been working in leading roles in several national and international research projects on mobile computing and communications, including EC projects BRAIN (IST-1999-10050) and MIND (IST-2000-28584). His research interests include mobile computing, wireless networking, distributed systems, and computer networks. He is the author of a number of scientific publications in these areas. He has made several contributions to IETF. He is member of the Finnish Society of Computer Science, ACM, ISOC, and ISOC Finland.

1 Introduction

Today mobile nodes (MNs) are often equipped with multiple radio interfaces to connect to access networks using diverse link technologies in order to support the best of connectivity, services quality, application needs and user preferences. In such a multi-access mobile environment, the connectivity of an MN may change very widely across access networks with different orders of bandwidth, latency and error characteristics during the lifetime of a connection. The switching between the access points of different type is known as a vertical handoff (Manner and Kojo, 2004).

A significant change in the access link characteristics easily affects the end-to-end path properties and thereby the behaviour of transport protocols. In the case of transmission control protocol (TCP) (Postel, 1981), the most widely used transport protocol in the internet, a vertical handoff may incur packet losses, intermittent connectivity, packet reordering and spurious retransmission timeouts (RTOs) resulting in unnecessary TCP congestion response or inefficient loss recovery that sacrifice TCP performance (Hansmann and Frank, 2003; Kim and Copeland, 2003; Gurtov and Korhonen, 2004; Kim and Copeland, 2004; Huang and Cai, 2005, Schütz et al., 2005; Daniel and Kojo, 2006; Sarolahti et al., 2006; Schütz et al., 2008).

When a vertical handoff occurs, the TCP sender adjusts its transmission rate and RTO estimate very slowly to the new end-to-end path as it learns the properties of the new path implicitly by probing it over several round trips. If the TCP layer is explicitly notified about the changes in the path properties, the TCP sender could decide whether the path characteristics have changed notably and react more timely and efficiently and possibly avoid false congestion responses. The TCP layer on the MN can be locally informed of the changes in the attached access link and its characteristics by using a cross-layer notification. However, the TCP layer at the other end of the connection is not aware of such changes. Therefore, introducing an explicit end-to-end indication is needed.

The earlier proposals in improving TCP with vertical handoff can be categorised into two main classes:

1 sender-based algorithms

2 receiver-based algorithms.

The sender-based algorithms (Daniel and Kojo, 2006; Sarolahti et al., 2006; Tsukamoto et al., 2006; Lin and Chang, 2007; Schütz et al., 2008) assume that the sender gets an explicit notification that the handoff has occurred and enhancements to the TCP sender algorithm are proposed. The receiver-based algorithms (Matsushita et al., 2007) modify the TCP receiver algorithm when the receiver gets the notification regarding the handoff. Hansmann and Frank (2003); Kim and Copeland (2003), Huang and Cai (2005) propose both TCP sender and receiver enhancements.

In this paper we make the following contributions:

1 We analyse TCP performance with a vertical handoff between access networks having a wide range of link bandwidth and delay to identify the various problems that affect TCP behaviour.

2 We deliver explicit link delay and bandwidth information to the TCP sender and by taking advantage of this information we develop further our cross-layer enhanced TCP algorithms to cover a wider set of changes in the access link characteristics.

3 We compare the performance of TCP enhanced by our algorithms with the regular TCP performance. We propose delivering information about access link characteristics in the mobility signaling messages from the MN to the correspondent node (CN). The simulation study is carried out with ns-2 network simulator (Network Simulator ns-2, 2005).

We demonstrate that our proposed enhancements are effective in avoiding spurious RTOs, reducing packet losses due to change in the capacity of the links, improving the link utilisation immediately after a disconnection and converging to the RTO value of the new end-to-end path quickly. As we are interested in studying the behaviour of TCP due to handoff we study how TCP behaves immediately after a handoff. As a performance index, we calculate the time taken to transfer (to get the acknowledgment) 100 new data packets through the new path after a handoff. With the proposed algorithms TCP performance is improved in many of the handoff scenarios and in some scenarios the improvement is more than a factor of two. Our proposed enhancements are conservative in nature and do not adversely affect the TCP performance when the cross-layer notification is unavailable.

The rest of the paper is organised as follows. Section 2 gives a brief description of the related work in this area. In Section 3 we introduce the simulation setup used in carrying out the vertical handoff experiments. In Section 4 we discuss our findings on the problems affecting TCP performance with a vertical handoff and in Section 5 we discuss these problems further and propose solutions to mitigate these problems. Section 6 evaluates the performance of the proposed TCP algorithms in various vertical handoff scenarios along with a comparison with TCP. In Section 7 we present our conclusions.

2 Related work

Stemm and Katz (1998) introduced the term vertical handoff in the context of wireless overlay networks. They defined vertical handoff as the switching between base stations which use different link level technologies in a wireless overlay network.

Handoffs can be divided into two categories based on the connectivity to the access router: break-before-make and make-before-break (Manner and Kojo, 2004). In break-before-make handoff, an MN is associated with only one access point at a time and the old connection breaks before the new connection is operational while in make-before-break handoff, the MN is associated with more than one access point at a time and it ends its connection to the old access router only after establishing the connection to the new one.

Hansmann and Frank (2003) identify packet reordering, segment burst due to delay difference, and packet losses due to a decrease in the access link bandwidth-delay product (BDP) as the main problems affecting TCP due to a vertical handoff. They propose a *nodupack* scheme for packet reordering which suppresses the transmission of dupacks during the handoff. The authors also propose to reduce the congestion window (*cwnd*) to overcome the packet losses due to a BDP change. However reducing only the *cwnd* may make TCP more aggressive by taking it to slow-start.

A TCP scheme for seamless vertical handoff (Kim and Copeland, 2003) introduces a handoff option (HO) in the TCP header to identify the beginning and end of a handoff. During a vertical handoff the TCP sender stops the retransmission timer, suspends the

data transfer and initiates a slow-start after the handoff. However with a make-before-break handoff, entering the slow-start phase unnecessarily retransmits many packets. In another paper by the same authors (Kim and Copeland, 2004), it is pointed out that a sudden change in RTT due to a handoff affects the TCP performance. The paper proposes that by resetting the retransmission timer after a handoff from a high-delay network to a low-delay network performance of TCP Reno can be improved.

The RFC 3708 on DSACK use (Blanton and Allman, 2004) states that undoing the incorrect congestion measures due to packet reordering can be taken only if we confirm that all retransmitted packets in a particular window are retransmitted unnecessarily. In vertical handoff scenarios where there is a significant change in delay (and bandwidth) of the paths involved in a handoff, restoring the old *cwnd* and *ssthresh* values may adversely affect the performance of TCP if the BDP of the new path is smaller than the BDP of the old path.

Eifel algorithm (Ludwig and Meyer, 2003) and F-RTO algorithm (Sarolahti and Kojo, 2005; Sarolahti et al., 2003) are two approaches to detect spurious RTOs. In vertical handoff scenarios, these algorithms are effective in avoiding the unnecessary retransmissions but not in congestion control response as the selection of a proper response is hard without additional information about the new path.

Overbuffering is proposed in Gurtov and Korhonen (2004) to mitigate the problems of a BDP change. However, this scheme is not easy to implement as the operators would need to know the bandwidth and delay of all the links on the end-to-end path in setting the buffer sizes and so overbuffering in all nodes along the path is hard to deploy.

A practical study on the performance of TCP with vertical handoff between GPRS and WLAN is presented in Chakravorty et al. (2004). This study identifies the delay in handoff as the cause of TCP to timeout often thereby affecting TCP performance. The high buffering in GPRS aggravates the performance of TCP as it inflates the round trip time (RTT) and retransmission timeout (RTO) values.

To solve the problem of spurious RTOs caused by the increase in RTT after a vertical handoff, both sender and receiver based enhancements are proposed in Huang and Cai (2005). The basic idea behind these schemes is to reduce the difference in RTT between the old link and the new link either by sending a few packets through the new slow link or by sending a few ACKs through the old fast link. In order to get good results, one should be able to determine a proper value for the duration of using the old or the new link for the above purpose.

Schütz et al. (2005) propose TCP retransmission trigger which causes TCP to attempt a retransmission when the connectivity is restored and it shows that this method is useful for paths with intermittent connectivity. However, there can be unnecessary retransmissions with this scheme in the case of a make-before-break handoff. Our earlier proposal *rxmt-immediate* (Daniel and Kojo, 2006) is more conservative than the retransmission trigger as retransmission takes place only if TCP sender is in RTO recovery. Schütz et al. (2008) propose an extension to TCP, *TCP response to connectivity change indications* (RLCI) in response to a lower-layer notification called connectivity change indications (CCI). A TCP sender receives the CCI either from its local stack or through a TCP option when there is a change in connectivity. CCI is taken as a signal to TCP to re-probe the network path to find the characteristics of the new path. The TCP sender may send an initial window of data on the new path and reset the congestion control state, RTT variables and RTO timer as recommended in RFC 2988 (Paxson and Allman, 2000) for a new connection. The *cwnd* should not be adjusted when the ACKs

for the packets delivered through the old link are received as they do not reflect the current path parameters. TCP timestamps option (Borman et al., 1992) may be used to distinguish the ACKs transmitted before or after the CCI. If a connection is stalled in an exponential backoff, TCP may retransmit the first unacknowledged segment.

Lin and Chang (2007) propose a vertical handoff-aware TCP (VA-TCP) where a TCP sender gets a notification from the MN regarding the handoff. During a vertical handoff when the TCP sender detects that a packet sent over the old access network is lost, it only retransmits the missing segment but does not invoke any congestion control actions. After the handoff, the TCP sender estimates the bandwidth and RTT using the packet-pair scheme (Keshav, 1991) and sets the *cwnd* and *ssthresh* to the BDP calculated using the bandwidth and the RTT estimates. A potential problem with this approach is that if a handoff occurs to a network with a smaller BDP continuing with the old *cwnd* even for a few RTTs during the packet-pair estimation may congest the network leading to a costly RTO recovery..

ACK pacing (Matsushita et al., 2007) is a receiver based mechanism to improve the performance of TCP with vertical handoffs. When a handoff decision is made, an MN calculates the BDP of the end-to-end path before and after the handoff. If the BDP of the new path is less than that of the old path, the TCP receiver sends duplicate ACKs until the transmission rate is reduced below the bandwidth of the new wireless access link. After the handoff the ACKs are sent at a rate depending on the bandwidth of the new link. However, the duplicate ACKs will force the sender to unnecessarily retransmit an already received packet. If the BDP of the new path is larger, multiple partial ACKs are sent until the transmission rate increases to the bandwidth of the new link. However, sending more than one ACK for each received packet is not advisable as malicious users can exploit it to increase the sending rate aggressively (Allman et al., 1999 and Savage et al., 1999).

Tsukamoto et al. (2006) address the problem of a large change in bandwidth of the access links before and after a vertical handoff and proposes two schemes to overcome this problem. In the first scheme, the TCP sender goes to slow-start as soon as the interface change is detected. However, going to slow-start may lead to inefficient utilisation of the wireless link after a handoff and also may unnecessarily retransmit the segments whose ACKs are delayed in the case of a make-before-break handoff from a low-delay to a high-delay link. In the second scheme known as bandwidth-aware scheme, TCP sender goes to slow-start after a vertical handoff and the bandwidth of the new path estimated using a single packet-pair (Keshav, 1991). The *ssthresh* is set to the BDP calculated using the bandwidth and the RTT of the new path. Simulation results show that bandwidth-aware scheme is capable of 80% utilisation of the available bandwidth.

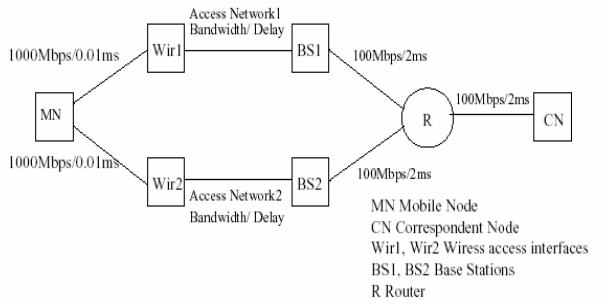
As there may be a significant change in link characteristics due to a vertical handoff, Sarolahti et al. (2006) study the suitability of Quick-Start (Sarolahti et al., 2007a) to set the *cwnd* and *ssthresh* after a vertical handoff. An explicit cross-layer handoff notification is employed to trigger quick-start when the handoff completes. Simulation results show that TCP performance is improved significantly by using quick-start after a vertical handoff.

The work reported in this paper is an extension of our earlier work (Daniel and Kojo, 2006). We present a comprehensive study of the problems of TCP in the presence of vertical handoff along with a detailed analysis of the algorithms and the results of the experiments.

3 Simulation setup

The simulation model we use in our experiments is shown in Figure 1. The MN is capable of switching between two wireless access interfaces, namely Wir1 and Wir2. Both Wir1 and Wir2 have dedicated base stations BS1 and BS2 that are connected to a common access router R which has a link to the CN. The delay and bandwidth of the fixed links are as shown in Figure 1. Our simulation model reflects the vertical handoff realistically by using the ns-2 routing features, i.e., by changing the route the packets take before and after a handoff. We assume that during a handoff, only the access links are changed and the rest of the path remains the same. When a handoff notification arrives, the route metric is changed to model a make-before-break handoff. To model a break-before-make handoff, an error model with a packet loss rate of 100% is applied to the old link and after the disconnection period, the route metric of the new link is set to a small value. In this model we can treat the packets from different wireless interfaces separately.

Figure 1 Network topology used in vertical handoff experiments



The nodes Wir1 and Wir2 are introduced to correctly model the break-before-make handoff. In ns-2 when an error model is applied, the packets are dropped at a node. Without Wir1/Wir2, if we apply the error model at BS1/BS2 the packets in transit from BS1/BS2 to MN will not be dropped. The link between MN and Wir1/Wir2 has 'infinite buffer' and 'no' propagation delay, i.e., Wir1 and Wir2 are local to MN and do not contribute to the delay of MN's path to the base station. In order to drop both data as well as ACKs during a break-before-make handoff the error model is applied in both directions of the old link.

4 Analysis of TCP behaviour with vertical handoff

In this section we discuss our findings on regular TCP performance with a vertical handoff based on simulations. The purpose of the simulation is to identify the problems of TCP in vertical handoff involving links of wide-ranging bandwidth and delay. The results are used as basis for developing the enhanced algorithms discussed in Section 5.

In order to study the effects of changes in link bandwidth and delay on TCP behaviour we categorise our experiments into the following three classes:

1 handoff between links which have the same bandwidth but different delay

2 handoff between links which have the same delay but different bandwidth

3 handoff between links which have the same BDP but bandwidth and delay differ.

The choice of the parameters for the bandwidth and delay cover the entire range of values that are of interest in typical handoff scenarios.

We use the TCP SACK algorithm implemented in the ns-2 simulator. There is a single TCP flow, for instance a file transfer, from the CN to the MN. The TCP packet size is 1500 bytes with the TCP/IP headers included. The router buffer size of each link is set to the BDP of the link if the BDP is greater than five packets; otherwise, it is set to five packets. A handoff can occur once in the lifetime of a TCP connection in any of the slow start, slow-start overshoot, fast retransmit/fast recovery or congestion avoidance phases. In our experiments a 20-second interval is chosen to cover all the phases of a TCP connection and a handoff can occur uniformly in any of the 200 points at 100 ms intervals. The duration of each test run includes the completion of the handoff occurring in the 20-second interval. Both make-before-break as well as break-before-make handoffs are examined. The disconnection period for break-before-make handoff is taken to be 500 ms. No link errors are modelled as we assume that the packet losses are either due to disconnection or congestion. This choice is made as the present study is devoted to the effect of vertical handoff on TCP.

In all the experiments, the parameter (bandwidth/delay) of either the old link or the new link is kept constant and we vary the parameters of the other link. As we are interested in studying the behaviour of TCP with a vertical handoff we study how TCP behaves immediately after the handoff. As a performance index, we calculate the time taken to transfer (to get the acknowledgment) 'n' new data packets through the new path after a handoff where n varies from 50 to 200. We report the results only for the case where n is 100 as the results we get are similar for all values of n. In all the performance graphs given in this paper, the x-axis shows the link parameters and the y-axis shows the lower quartile, median, and upper quartile of the time (in seconds) to transfer 100 packets after the handoff.

4.1 TCP behaviour due to changes in delay

The aim of these experiments is to study the effect on TCP of a change in the access link delay arising from a vertical handoff. We vary the delay of one of the links involved in a handoff while the delay of the other link is kept fixed at 300 ms. The varying link delays are 150 ms, 75 ms, 37 ms, 18 ms, 9 ms and 1 ms so that the ratio between the delays of the two links is two, four, eight, 16, 32 and 300 respectively. This range is wide enough to accommodate the majority of different access links deployed at present. These experiments are repeated for link bandwidths of 200 Kbps, 1600 Kbps and 6400 Kbps. As the link delay is the varying parameter in all the experiments, we study the behaviour of TCP with handoff from a low-delay link to high-delay link and high-delay link to low-delay link separately.

Figure 2 shows the transfer time for make-before-break and break-before-make handoffs on 6400 Kbps and 200 Kbps links as the delay of the old link varies from 150 ms to 1 ms while the delay of the new link is fixed at 300 ms. The main problem that affects the performance of TCP in a make-before-break handoff from a low-delay link to

a high-delay link are the occurrence of spurious RTOs and the unnecessary congestion control actions associated with it. As a result of a spurious RTO, the TCP sender retransmits packets unnecessarily and decreases the sending rate by reducing the *cwnd* and *ssthresh.*

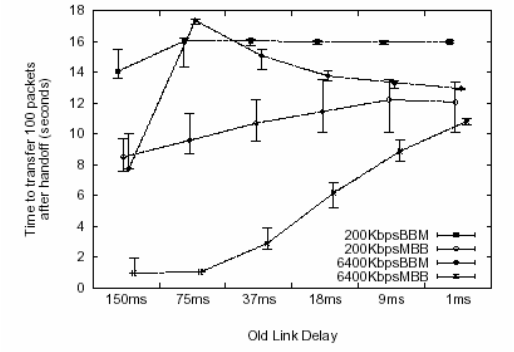
Given that the delay increase is significant, the small RTO value based on the measurements over the low-delay path causes the TCP retransmission timer to expire spuriously as the ACKs take the high-delay link after the handoff resulting in a significant increase in RTT. Typically no packet losses occur during a make-before-break handoff. Hence when the TCP sender times out spuriously, it retransmits a full window of packets unnecessarily and continues in congestion avoidance with reduced *cwnd* resulting in performance degradation.

As can be seen from Figure 2 due to the adverse effect of spurious RTOs, the performance penalty with the make-before-break handoff becomes more severe with the increase in the ratio of the delays of the old and the new links. We observe that for 1600 Kbps and 6400 Kbps links with make-before-break handoff, spurious RTOs occur in more than 85% of the handoff points when the delay of the new link is at least eight times the delay of the old link and they occur in less than 20% of the handoff points when this ratio is less than eight. For low-bandwidth links, such as 200 Kbps links, serialisation delay reduces the ratio between the old and the new link delays thereby reducing the occurrence of spurious RTOs. Our experiments with 200 Kbps links show that the spurious RTOs occur only in 10–40% of the handoff points, increasing with the decrease in the old link delay.

With a break-before-make handoff the connectivity is lost for some period of time and resumes after the handoff completes. During this disconnection period the retransmission timer may expire several times, each time doubling the RTO value. When the connectivity is resumed, the TCP sender needs to wait until the retransmission timer expires again before attempting another retransmission. This unused connection time delays the start of the recovery from lost packets. If more than one timeout has occurred, i.e., a retransmission is lost, *ssthresh* value is further reduced. However, the retransmission is lost due to disconnection not due to congestion, making this reduction of *ssthresh* unnecessary.

In Figure 2 we can see that for the 6400 Kbps links, the break-before-make handoff from a 75 ms link to a 300 ms link shows a sharp increase in transfer time. Due to the relatively high BDP of the 75 ms link (80 packets) a large number of packets are lost due to disconnection. This typically requires an RTO recovery. The retransmission timer expires once during the disconnection period of 500 ms and the TCP sender doubles the RTO value. Another RTO is required to recover the losses and *ssthresh* is reduced to two after this RTO. This reduction in *ssthresh* is the main reason for the long transfer time. Even if the link becomes operational before the second RTO, TCP will not retransmit the lost segment until the second RTO occurs. Here we observe an unused connection time (100 ms to 300 ms) after the handoff completes which further increases the transfer time. For the handoff from the 150 ms link, the retransmission timer will not expire during the disconnection period of 500 ms and the lost packets are recovered by a single RTO recovery and *ssthresh* is reduced just once. With smaller link delays (i.e., 9 ms and 1 ms) the link BDP is small and only a small number of packets are lost during the disconnection. For the break-before-make handoff between 200 Kbps links as the link BDP is less than five packets it never results in significant number of packet losses and the transfer time remains roughly the same in all cases.

Figure 2 Handoff from a low-delay to a high-delay link with a fixed bandwidth (200 Kbps, 6400 Kbps)



Note: Transfer time for 100 packets after a make-before-break (MBB) and break-before-make (BBM) handoff with varying delays of the old link. The delay of the new link is fixed at 300 ms.

With a make-before-break handoff from a high-delay link to a low-delay link, the main problems of TCP are due to

1 the decrease in the link BDP after a handoff resulting in congestion-related packet losses

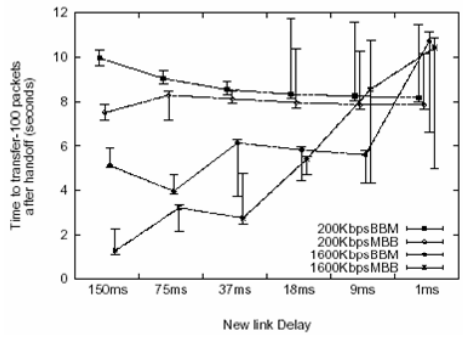
2 the ACKs arriving through the new link triggering more packets to be sent to the low BDP link resulting in packet losses

3 the slow convergence of RTO to the new path delay.

Figure 3 shows the make-before-break and break-before-make handoffs from a 300 ms delay link to a new link with varying link delays when the link bandwidth is 200 Kbps/1600 Kbps. In the case of 1600 Kbps links, when a make-before-break handoff occurs from a 300 ms link to a low-delay link, many packets are dropped due to the large decrease in BDP which requires RTO recovery. For the low link delays (delay from 18 ms down to 1 ms), after a make-before-break handoff, as the ACKs arrive through the new fast link the sender injects packets quickly to the new low BDP link which further congests the link. The initial packet losses due to the decrease in BDP and the packet losses due to the high sending rate after the handoff cause a series of RTOs. This accounts for the transfer time being nearly the same or greater than the corresponding transfer time for break-before-make handoff when the new link delay decreases from 18 ms to 1 ms. The large difference between the median and the quartiles for the make-before-break handoff when the delay of the new link is either 9 ms or 1 ms link is due to the variable number of RTOs required for the loss recovery. The high buffering in the old high BDP link inflates the RTO value and invoking RTO recovery takes a long time due to the slow convergence of the RTO to the new path values. We observe a similar TCP behaviour for the handoff between 6400 Kbps links. In the case of 200 Kbps links, the decrease in BDP due to handoff is within five packets for all the delay values of the new link. As a result the graph for make-before-break handoff with a 200 Kbps link is

nearly constant. In a break-before-make handoff from a high delay link to a low-delay link, all packets sent during the disconnection are lost in addition to packets lost due to the decrease in BDP. This accounts for the increase in transfer time of a break-before-make handoff compared to that of a make-before-break handoff in a similar scenario.

Figure 3 Handoff from a high-delay link to a low-delay link with fixed bandwidth (200 Kbps, 1600 Kbps)



Note: Transfer time for 100 packets after a make-before-break (MBB) and break-before-make (BBM) handoff with varying delays of the new link. The delay of the old link is fixed at 300 ms.

Packet reordering is observed when a make-before-break handoff occurs from a high-delay to a low-delay link as packets with higher sequence numbers traversing the new low-delay link arrive at the receiver earlier than the packets sent through the old high-delay link before handoff. A false fast retransmission will be triggered only when at least a *dupthresh* number (usually three) of out-of-order segments for each in-order segment are received. As the bandwidth remains the same before and after the handoff, sufficient dupacks may not be generated to trigger a false fast retransmission.

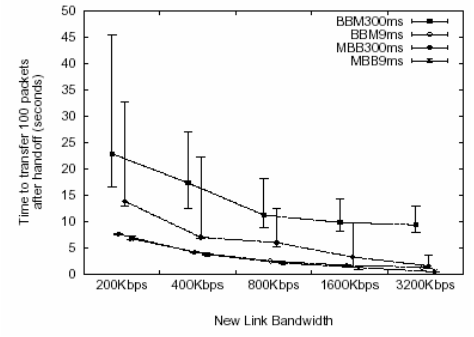
4.2 TCP behaviour due to changes in bandwidth

In this set of experiments, we vary the bandwidth of the links involved in the handoff while keeping the delay of the links constant. The bandwidth of one of the links is varied while the bandwidth of the other link is kept fixed at 6400 Kbps. The varying link bandwidths are 200 Kbps, 400 Kbps, 800 Kbps, 1600 Kbps and 3200 Kbps. The experiments are conducted for the link delays of 300 ms, 75 ms, 9 ms and 1 ms. As the link bandwidth is the only variable, we study the behaviour of TCP in a handoff from a low-bandwidth link to a high-bandwidth link and from a high-bandwidth link to a low-bandwidth link separately**.**

Figure 4 shows the transfer times for make-before-break and break-before-make handoffs when the bandwidth of the old link is fixed at 6400 Kbps and the new link bandwidth is varied. The major problem affecting TCP here is the packet losses due to decrease in BDP. For the break-before-make handoff, the recovery of the lost packets due to disconnection increases the transfer time compared to that of the make-before-break

handoff. For the 6400 Kbps/300 ms link, the slow-start overshoot starts around 9.3 seconds resulting in large packet losses due to the high BDP of the link (320 packets) which leads to an RTO recovery. It can be seen in Figure 4 that for links with 300 ms delay the BDP decrease is maximum when the bandwidth decreases from 6400 Kbps to 200 Kbps and the transfer time shows maximum increase. When a handoff from 6400 Kbps to 200 Kbps occurs during the slow-start recovery, TCP needs a series of RTOs to recover the lost packets as there is a significant decrease in the BDP of the new link (from 320 packets to ten packets). This accounts for the high third quartile value (almost double the median) of the transfer time. The high transfer time decreases with increase in bandwidth of the new link.

Figure 4 Handoff from a high-bandwidth link to a low-bandwidth link with fixed delay (9 ms, 300 ms)



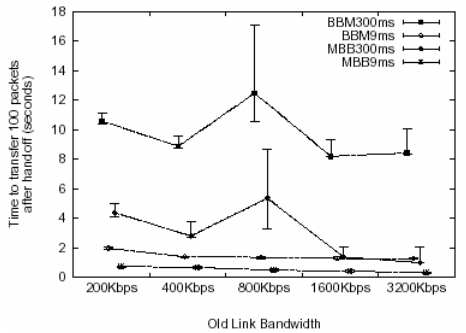
Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff between same delay links with varying new link bandwidth and the bandwidth of the old link fixed at 6400 Kbps.

In the make-before-break handoff between low-delay links (9 ms and 1 ms), the increase in serialisation delay due to the decrease in bandwidth after a handoff may result in spurious RTOs. The major problems affecting TCP here are the decrease in BDP and spurious RTOs. For higher delay links (i.e., 300 ms link), the serialisation delay adds little to the total delay and no spurious RTOs are observed.

Figure 5 shows the transfer time for the make-before-break and break-before-make handoffs between 300 ms and 9 ms links when the handoff occurs from a low-bandwidth link to a high bandwidth bandwidth link. The bandwidth of the old link is varied and the new link bandwidth is fixed at 6400 Kbps. The main problem affecting TCP here is its inability to efficiently utilise the high bandwidth available after a handoff. We can see in Figure 5 that the transfer time between 300 ms delay links depends mainly on the bandwidth of the old link even though the new link bandwidth/delay is 6400 Kbps/300 ms in all the cases and the new link has a higher BDP than the old link. For the handoffs occurring during or after the slow-start overshoot, the reduced *cwnd* and *ssthresh* of the old path decrease the TCP sending rate even though a high BDP link is available after the handoff. In the worst affected case of 800 Kbps link, the slow-start overshoot starts around 6.5 seconds and approximately 120 packets are lost. The first lost packet is retransmitted in fast retransmit but the ACK gets delayed resulting in a spurious RTO. As

the link is already congested due to slow-start overshoot, some retransmitted packets are also lost resulting in another timeout. The consequent reduction in *cwnd* and *ssthresh* values further reduces the sending rate for handoffs occurring after the slow-start overshoot. There is an additional burden to the recovery in the break-before-make handoff owing to packet losses due to disconnection.

Figure 5 Handoff from a low-bandwidth link to a high-bandwidth link with fixed delay (9 ms, 300 ms)



Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff between same delay links with varying old link bandwidth. The bandwidth of the new link fixed at 6400 Kbps.

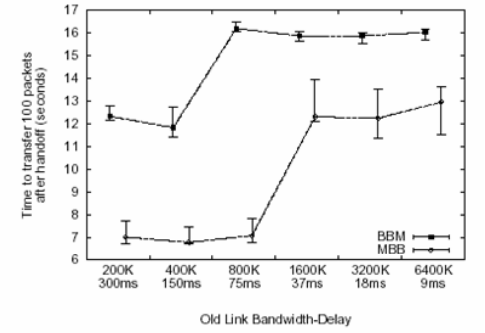
TCP behaviour for make-before-break handoff between 9 ms delay links is similar to the behaviour we described for 300 ms links. In the case of break-before-make handoff, TCP will be in RTO recovery during the disconnection period and another RTO is required to recover the losses. As explained in Section 4.2 the unused connection time and the *ssthresh* reduction are the main problems due disconnection. The high RTO value of the low-bandwidth links increases the unused connection time resulting in increased transfer time. For 200 Kbps links, the unused connection time is about 700 ms and as the bandwidth increases the unused connection time decreases. As the RTO value gets clamped to the *minrto*, the unused connection time for higher bandwidths (from 800 Kbps) is 100 ms. As the *ssthresh* reduction is not significant here as the BDP of the old link for all the bandwidths used in the experiments are less than or equal to five.

4.3 TCP behaviour due to changes in bandwidth and delay for fixed bandwidth-delay product (BDP)

In this set of experiments, the bandwidth and delay of the links involved in a handoff are different while their BDP remains unchanged. We vary the bandwidth and delay of one of the links involved in the handoff while the bandwidth and delay of the other link are kept fixed. We have two sets of fixed bandwidth/delay values namely, 200 Kbps/300 ms and 6400 Kbps/9 ms. For the varying link, the bandwidth/delay combinations are 200 Kbps/300 ms, 400 Kbps/150 ms, 800 Kbps/75 ms, 1600 Kbps/37 ms, 3200 Kbps/18 ms and 6400 Kbps/9 ms. With these combinations, both old and new access links have a

BDP of ten packets. We perform the experiments for a handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link and vice versa.

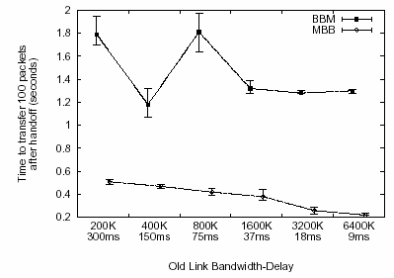
Figure 6 Handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link with fixed BDP



Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link and the new link fixed at 200 Kbps/300 ms.

Figure 6 shows that a significant decrease in bandwidth/increase in delay due to a handoff increases the transfer time for both make-before-break and break-before-make handoffs. Here we fix the new link bandwidth/delay at 200 Kbps/300 ms while varying the bandwidth and delay of the old link. When there is a significant increase in delay after a make-before-break handoff, TCP suffers from spurious RTOs whereas in a break-before-make handoff, unused connection time and the *ssthresh* reduction are the main problems. Spurious RTOs occur in more than 90% of the handoff points when the ratio of change in bandwidth (or delay) is at least eight.

Figure 7 Handoff from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link with fixed BDP



Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link and the bandwidth and delay of new link fixed at 6400 Kbps/9 ms.

When the old link bandwidth is 1600 Kbps or higher, many packets are lost in a make-before-break handoff due to the bursty transmission caused by the arrival of late ACKs at the high rate of the old link resulting in heavy congestion on the new low-bandwidth link. In most of the cases, recovery needs one or more RTOs and the reduction in the sending rate is drastically affected by the reduced *ssthresh* and *cwnd.*

Figure 7 shows the performance of make-before-break and break-before-make handoffs from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link when the new link is fixed at 6400 Kbps/9 ms. Packet reordering is a problem affecting TCP when there is a significant reduction in delay after a make-before-break handoff. In the case of break-before-make handoff with a high delay old link (for 200 Kbps/300 ms and 400 Kbps /150 ms links), the high RTO value prolongs the start of the RTO recovery. For links with lower delay (from 75 ms to lower link delays) the increase in transfer time is mainly due to the occurrence of multiple RTOs and the resulting reduction in *ssthresh*. The maximum unused connection time and the reduction in *ssthresh* account for the peak in the transfer time for the break-before-make handoff from 800 Kbps/75 ms link to 6400 Kbps/9ms link in Figure 7.

5 Cross-layer enhanced TCP algorithms

In this section, we propose solutions to mitigate the problems of TCP described in Section 4. Our algorithms described here take a conservative approach in setting the TCP congestion control parameters. We elaborate further the problems with TCP behaviour in vertical handoff scenarios while we discuss the proposed enhancements to the TCP sender algorithm. The baseline TCP used in our experiments is the TCP SACK algorithm implemented in ns-2 simulator and we refer to it as regular TCP. The enhancements are invoked upon arrival of a handoff notification from the lower layer. Here we assume that the MN sends the handoff notification to the TCP sender at the CN, including an estimate of the bandwidth and delay of the two access links involved in the handoff.

5.1 Cross-layer notifications

A number of mechanisms to support host mobility in IP networks have been developed. They include both basic protocol support for IP mobility such as Perkins (2002), Johnson et al. (2004), Henderson (2007) and various enhancements to reduce packet loss as well as the amount and latency of signalling, for example Koodli (2005), Soliman et al. (2005), El Malki (2007). All these mobility management mechanisms aim at hiding the host mobility from the layers above IP. However, this is not a viable approach if the implications of the mobility interact with the segment delivery at the transport layer. A vertical handoff that results in significant changes in end-to-end path properties cannot be totally hidden from the transport layer even if the handoff latency is reduced to minimum and no packets are dropped. Therefore it would be advisable to explicitly notify the transport layer of any significant changes in path properties.

Cross-layer notifications have been shown to be beneficial to TCP when path characteristics change widely due to a vertical handoff (Schütz et al., 2005; Daniel and Kojo, 2006; Sarolahti et al., 2006). During a vertical handoff, the TCP layer at the MN can be locally notified about the changes in the path characteristics at the time the handoff is executed. This information regarding the path characteristics can be sent to the

TCP layer at the CN, for example, as TCP options (Schütz et al., 2008), or along with the mobility registration message such as the Binding Update message in mobile IPv6 (Johnson et al., 2004) or with the readdress packet in HIP (Moskowitz and Nikander, 2006) to be further forwarded to the TCP layer. A TCP sender can take the notifications about the characteristics of the network path and adjust the congestion control parameters and RTO estimate so as to adapt to the new path in an efficient and timely manner thereby improving transport performance. In this paper we suggest delivering the notifications piggybacked in the mobility signalling messages so that the notifications can be delivered to the TCP layer exactly when the handoff completes.

5.2 TCP enhancements to avoid spurious RTOs

A more detailed analysis of results discussed in Section 4 exposes that spurious RTOs can occur mainly due to the following conditions when a make-before-break handoff occurs:

1 there is a significant increase in link delay after the handoff

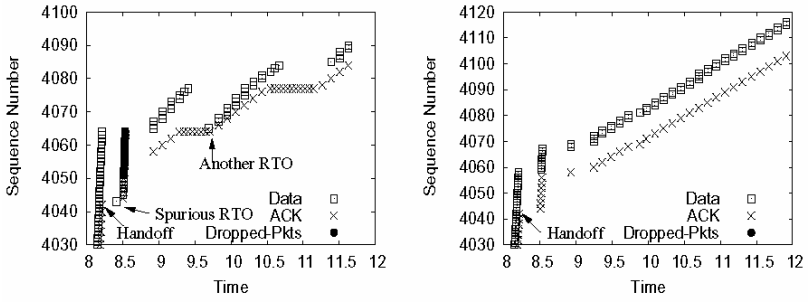
2 TCP is in fast recovery when the handoff occurs

3 TCP enters fast recovery after the occurrence of the handoff but before the ACKs for all packets sent before the handoff are received.

Next we discuss these conditions in detail.

Case 1

Figure 8 Comparison of regular TCP (a) and enhanced TCP (b): make-before-break handoff at 8.2s from a 6400Kbps/9ms link to a 200Kbps/300ms link



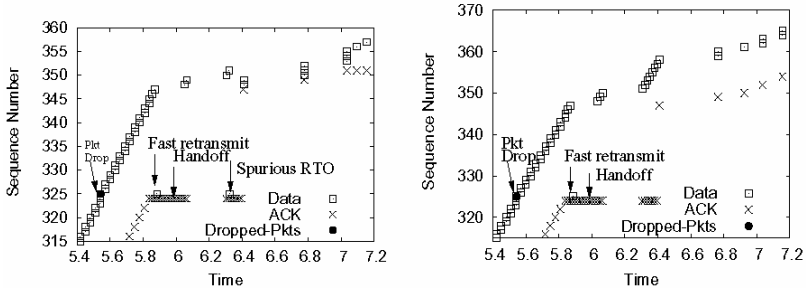
In Sections 4.2 and 4.3, we observed that spurious RTOs occur in more than 85–90% of the handoff points when the delay of the new link is at least eight times that of the old link. A typical scenario giving the time-sequence graph of a make-before-break handoff from a 6400 Kbps/9ms link to a 200 Kbps/300 ms link is shown in Figure 8a. We see that handoff occurs at 8.2 seconds after the beginning of a TCP connection, causing a sudden increase in RTT as the ACKs start taking the new high-delay link. A spurious RTO occurs at 8.4 seconds and the TCP retransmits the first unacknowledged segment. The

late ACKs for the segments sent before the handoff, start arriving at 8.5 seconds triggering a retransmission of a full window of 21 segments unnecessarily. As the late ACKs arrive back-to-back roughly at the line rate of the old high-bandwidth link, unnecessary retransmissions are triggered at a rate which far exceeds the capacity of the new link. Therefore the new link is congested soon. The late ACKs for data segments that took the new path after the handoff start arriving at 8.8 seconds, triggering transmission of new data segments. These segments enter the router queue in front of the new link which is filled with unnecessary retransmissions and experience a long queuing delay in addition to the high delay of the new link. As the RTO estimate converges very slowly to the long RTT of the new path, the total delay for these segments exceeds the current RTO value resulting in another spurious RTO at 9.7 seconds followed by unnecessary retransmission of the current window again.

Case 2

Figure 9a shows an example where TCP is in fast recovery when a make-before-break handoff occurs from a 800 Kbps/75 ms link to a 200 Kbps/300 ms link. The TCP sender congests the old link and fast retransmits at 5.9 seconds. The retransmitted segment enters a long queue of the bottleneck router. After this a handoff occurs at 6.0 seconds. The retransmitted segment arrives at the TCP receiver after experiencing a relatively long queuing delay and triggers an ACK that takes the new high delay link, adding further delay in the arrival of the ACK. This causes a spurious RTO at 6.3 seconds. This condition arises even when the delay of the new link is only four times the delay of the old link, which alone is not enough to cause a spurious RTO.

Figure 9 Comparison of regular TCP (a) and enhanced TCP (b): make-before-break hand-off at 6.0s from a 800Kbps/75ms link to a 200Kbps/300ms link

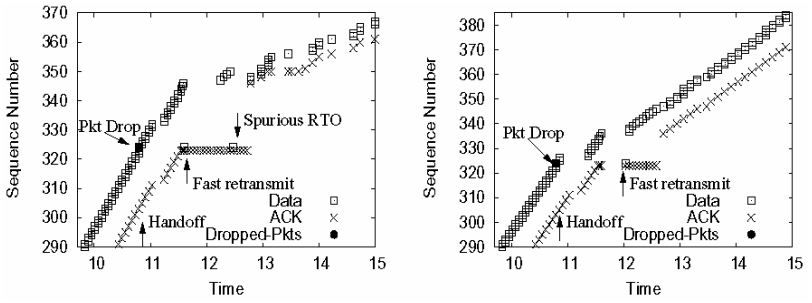


Case 3

Here the TCP sender enters fast recovery after a make-before-break handoff. This situation is illustrated in Figure 10a. The handoff is from a 400 Kbps/150 ms link to a 200 Kbps/300 ms link, where the ratio of change in delay is only two. The sender has already congested the old link after which a handoff occurs at 10.9 seconds. The TCP sender, unaware of the handoff, continues transmitting new segments clocked by the ACKs arriving roughly at the line rate of the old link and thereby quickly filling the router queue

in front of the new link. When the third dupack arrives at 11.6 seconds, indicating a packet loss on the old link, the TCP sender fast retransmits the lost segment. However, the queuing delay on the new link delays the delivery of the fast retransmitted segment and spurious RTO occurs at 12.4 seconds.

Figure 10 Comparison of regular TCP (a) and enhanced TCP (b): make-before-break handoff at 10.9s from a 400Kbps/150ms link to a 200Kbps/300ms link



In addition to the cases discussed above there are scenarios where queuing delay increases due to the increase in link serialisation delay resulting in spurious RTOs as the ACKs get delayed. This situation arises in make-before-break handoffs from a high-bandwidth link to a low-bandwidth link when TCP is either in fast recovery before a handoff or TCP enters fast recovery after a handoff. As the effect of serialisation delay is conspicuous for low delay links, these accounts for the occurrence of spurious RTOs for a make-before-break handoff from a high-bandwidth link to a low-bandwidth link described in Section 4.2 for the same delay link experiments (9 ms and 1 ms).

To avoid the occurrence of spurious RTOs we calculate the following parameters which are used in the enhanced TCP sender algorithm. In a make-before-break handoff where neither the data segment nor its ACK is lost, we may consider the data segment sent just before handoff traversing the old link with its ACK traversing the new link. The minimum RTT of this data segment-ACK pair can be calculated by the following formula taking into account only the access links.

$$RTT_{old\_newlink} = D_{oldlink} + SDpkt_{oldlink} + D_{newlink} + Dack_{newlink}$$

where

$SDpkt_{oldlink}$ - serialisation delay for a data packet on the old link

$SDack_{newlink}$ - serialisation delay for an ACK on the new link

$D_{oldlink}$ - propagation delay for the old link

$D_{newlink}$ - propagation delay for the new link

The RTT of the data segment-ACK pair traversing the new link is calculated based on the new access link delays:

$$RTT_{newlink} = 2 \cdot D_{newlink} + SDpkt_{newlink} + SDack_{newlink}$$

The new link BDP is calculated as follows:

$$BDP_{newlink} = BW_{newlink} \cdot RTT_{newlink}$$

where $BW_{newlink}$ - bandwidth of the new link.

The proposed algorithm to avoid spurious RTOs is given in Figure 11. The algorithm takes into account all three cases discussed above and it is invoked on the arrival of a handoff notification indicating an increase in the link delay. Here we calculate the *minrto* based on the new access link delay and update the RTO timer immediately so that the new *minrto* comes into effect. Thus any change in the delay of the end-to-end path will be better reflected in the RTO calculation. It, however, takes effect only if the TCP sender is not already in RTO recovery when the notification arrives.

Figure 11 Algorithm A1 to avoid spurious RTOs.

```
When handoff notification arrives indicating
an increase in delay / decrease in bandwidth:
If (TCP not in RTO recovery)
  Save minrto
   /* Case 1 and Case 2 */
    If ((RTT_old_newlink > currentRTO) OR
        (TCP is already in Fast Recovery))
      minrto = RTT_newlink + FlightSize/BW_oldlink + 200ms
      Update RTO timer
   /* Case 3 */
    If (TCP enters Fast Recovery before all packets sent
        before handoff are ACKed)
      minrto = RTT_newlink + FlightSize/BW_newlink + 200ms
      Update RTO timer
   /* Case 1, Case 2 and Case 3 */
  If (there is an increase in link delay after handoff)
    When all packets sent before handoff are ACKed
      Initialize RTT variables as for a new connection
    When ACK for a new data segment arrives
      Update the RTT variables
  Restore minrto
```

Regarding Case 1 and Case 2, the spurious RTO occurs either due to a significant increase in the link delay or due to queueing delay of the old link. In order to address Case 1, the TCP sender checks if the $RTT_{old\_newlink}$ (the minimum estimated RTT for a data segment traversing the old link and its ACK taking the new link) is greater than the current RTO. For Case 2 the algorithm checks if the TCP is already in fast recovery when a handoff notification arrives. If the TCP sender is in fast recovery, the queuing delay in the old link may increase the effective RTT for the fast retransmitted segments beyond the current RTO value even though the $RTT_{old\_newlink}$ is not larger than the current RTO. If any of these conditions is true, we set the *minrto* to the sum of the $RTT_{old\_newlink}$, an estimate for the queuing delay in the old link and a rough estimate of the rest-of-the path delay (here taken as 200 ms which is the default *minrto* value used in ns-2 and in many

real TCP implementations). Strictly speaking, instead of $RTT_{newlink}$ we can use $RTT_{old\_newlink}$ but here we are slightly overestimating the new *minrto* value.

In order to address Case 3 the TCP sender sets the *minrto* to the sum of the $RTT_{newlink}$, an estimate for the queuing delay in the new link and a rough estimate of the rest-of-the path delay.

In addition to setting the *minrto* value, we update the RTO timer immediately to allow the new *minrto* value to take effect as soon as possible. When the ACK for all the segments sent before the handoff has been received, we initialise the RTT variables and the RTO timer as recommended for a new connection in RFC 2988 (Paxson and Allman, 2000) provided there is an increase in the link delay after the handoff. The RTT variables are updated immediately upon the arrival of the first valid ACK and the *minrto* is restored to its default value.

The effectiveness of the algorithm in avoiding spurious RTOs can be seen in Figures 8b, 9b and 10b against the corresponding cases shown in Figures 8a, 9a and 10a for regular TCP.

Figures 8a and 8b represent the behaviour of TCP with and without the algorithm A1 for Case 1. The ACK for the first outstanding segment arrives after 337 ms but the RTO value of the regular TCP at handoff is only 200 ms resulting in spurious RTO. Algorithm A1 calculates the *minrto* to be 552 ms, sets RTO to this value, thereby avoiding the occurrence of spurious RTO.

A comparison of Figures 9a and 9b shows the effect of algorithm A1 for Case 2. We see that TCP is in fast recovery when the handoff occurs at 6.0 seconds. There are 24 segments outstanding at that time. The arrival of the ACK for the fast retransmitted segment takes 529 ms which is more than the RTO value of the regular TCP (440 ms) at the handoff and the RTO timer expires spuriously before the arrival of the ACK. The use of algorithm A1 results in the *minrto* value of 1220 ms, thereby avoiding spurious RTOs.

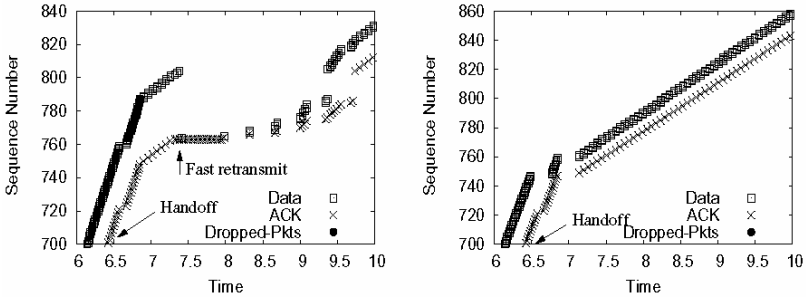
For Case 3, Figure 10a shows that there are 22 segments outstanding when TCP fast retransmits. The arrival of the ACK for the fast retransmitted packet takes 1.17 seconds which is more than the RTO value of the regular TCP (850 ms) at that point and spurious RTO occurs due to the late arrival of the ACK. Using the algorithm A1, the *minrto* is calculated to be 2.12 seconds and the resulting larger value of RTO is effective in avoiding the spurious RTO as can be seen in Figure 10b.

5.3 TCP enhancements to minimise congestion-related packet losses

Congestion-related packet losses may occur due to a handoff occurring from a high-BDP path to a low-BDP path. The BDP of the bottleneck link determines the minimum size of the TCP window that may fully utilise the bottleneck link. The congestion point of the bottleneck link is determined by the BDP of the link, the router queue size in front of the link and the number of packets in flight elsewhere on the end-to-end path.

Figure 12a shows a make-before-break handoff from 1600 Kbps/75 ms link to 400 Kbps/150 ms link. The old link BDP is 20 packets while the new link BDP is ten packets. When a handoff occurs at 6.5 seconds TCP continues to inject packets to the new link at the previous rate and several packets starting from sequence number 764 are lost. On the new high-delay link it takes about 2.5 seconds for the TCP sender to recover the lost packets and adapt to the sending rate of the new link.

Figure 12 Comparison of regular TCP (a) and enhanced TCP (b): make-before-break handoff at 6.5s from a 1600Kbps/75 ms link to a 400 Kbps 150 ms link



If the *FlightSize* at the time of handoff is greater than the buffering capacity of the new link, packet losses due to congestion may occur after the handoff. Therefore we check whether the *FlightSize* exceeds the estimated buffering capacity of the new link. We assume that the router queue size in front of the access link equals the BDP of the link so that the link has a total buffering capacity of twice the link BDP. The total buffer capacity of the end-to-end path is likely to be larger than this estimate allowing some slack in the estimate. In order to avoid the underutilisation of the new access link we reduce the congestion window (*cwnd*) and the slow-start threshold (*ssthresh*) to the BDP of the new link. The algorithm for reducing congestion-related packet losses is given in Figure 13 and is invoked if TCP sender is not in RTO recovery. A flag, *cwnd_reduced*, is set to one so that *cwnd* is not reduced further if TCP enters fast recovery to recover lost packets sent before the handoff. This flag is cleared when all packets sent before handoff is ACKed.

Figure 13 Algorithm A2 to reduce congestion-related packet losses

When handoff notification arrives
If ((TCP not in RTO recovery)
  If ($FlightSize > 2 * BDP_{newlink}$)
    $cwnd = max(2, BDP_{newlink})$
    $ssthresh = cwnd$
    $cwnd\_reduced = 1$

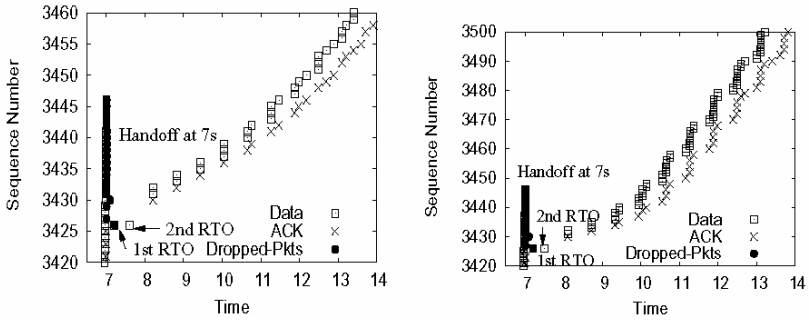
A comparison of Figure 12a and Figure 12b shows the effect of the algorithm A2. The *FlightSize* when the handoff notification arrives is 39 packets whereas the BDP of the new link is only 11 packets. As this *FlightSize* is larger than twice the BDP of the new link, the algorithm A2 sets the *ssthresh* and *cwnd* to the BDP of the new link which effectively avoids the congestion-related losses and allows the TCP sender to smoothly continue data transmission over the new link.

5.4 TCP enhancements to reduce the effect of disconnection

The main problems of TCP in a break-before-make handoff are excessive *ssthresh* reduction, packet losses and unused connection time. A typical break-before-make scenario shown in Figure 14a illustrates these problems. It shows a break-before-make

handoff between 6400 Kbps links with link delay changing from 9 ms to 300ms in a handoff. The handoff occurs at 7.0 seconds and there are 20 outstanding packets at that time. All the packets from sequence number 3427 to 3446 except 3428 and 3430 are lost due to the disconnection arising from the break-before-make handoff. Although packet 3426 was received, the TCP receiver did not send the ACK for it immediately because of the delayed ACK feature of TCP. In this scenario, we see that the first retransmission timeout occurs at 7.2 seconds and the retransmission of packet 3426 is lost during the disconnection period of 500 ms. A second retransmission of packet 3426 at 7.6 seconds reaches the receiver. After the first retransmission, the *ssthresh* is reduced to half of the *FlightSize* (ten packets) and after the second retransmission the *ssthresh* is further reduced to two. The loss recovery carried out in congestion avoidance takes a very long time over the high-delay link even though the BDP of the new link is very high (320 packets). The unused connection time is about 100 ms in this scenario.

Figure 14 Comparison of regular TCP (a) and enhanced TCP (b): break-before-make handoff at 7.0 s from a 9 ms link to a 300 ms link with identical bandwidth of 6400 Kbps



Note: Disconnection period is 500ms.

Algorithm A3 shown in Figure 15 is used to reduce the unused connection time and to avoid the unnecessary *ssthresh* reduction. Upon the first expiry of the retransmission timer for a given TCP segment, the TCP sender reduces *cwnd* and *ssthresh* as usual and then saves the reduced *ssthresh* value. If the TCP sender is already in RTO recovery when the handoff notification arrives, the TCP sender immediately retransmits the first unacknowledged packet and restores the saved value of *ssthresh.*

The effect of algorithm A3 in reducing the unused connection time and avoiding *ssthresh* reduction can be seen by comparing Figure 14a and Figure 14b. The improved performance of algorithm A3 is mostly due to the restored *ssthresh* value. With immediate retransmission in algorithm A3, the retransmission occurs at 7.5 seconds as soon as the connection is restored and the unused connection time of 100 ms in the regular TCP is eliminated. However, the effect of unused connection time becomes more visible for longer disconnection periods that allow RTO to backoff to a large value.

Figure 15 Algorithm A3 to reduce the unused connection time and to restore the *ssthresh*

```
On the first expiration of RTO:
    Reduce cwnd and ssthresh as usual
    Save ssthresh
When handoff notification arrives:
    If (TCP in RTO recovery)
      Retransmit the first unacknowledged packet
      Restore ssthresh
      If there is a significant change in delay
        Initialize RTT variables as for a new connection
        When ACK for new data arrives
          Update RTT variables
```

5.5 TCP enhancements for a fast convergence of RTO

In a handoff from a high-delay link to a low-delay link, the RTO value may be very high compared to the new end-to-end RTT. The RTO may be even higher due to the queuing delay, if the old link has a high BDP. After a handoff, the RTO will converge to the RTT of the low-delay path very slowly. This convergence is outstandingly slow when the RTT variables are updated only once in an RTT (Paxson and Allman, 2000). The high RTO value delays the timeout recovery unnecessarily if an RTO recovery is needed relatively soon after a handoff. The algorithm A4 given in Figure 16 helps TCP to converge faster to the new RTO value by initialising the RTT variables and updating the RTT variables again immediately when an ACK for a data segment sent over the new path arrives to reflect the end-to-end delay of the new path.

Figure 16 Algorithm A4 for fast convergence of RTO

```
When handoff notification arrives indicating
a decrease in delay:
      If(TCP is not in RTO recovery)
        Wait till the segments sent before
        handoff have been ACKed
        Initialize RTT variables as for a new connection
        When ACK for a new data segment arrives
          Update RTT variables
```

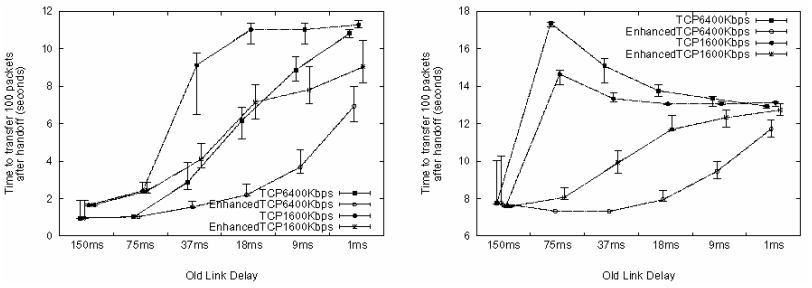
We designate the TCP sender algorithm incorporating the enhancements given by algorithms A1, A2, A3 and A4 as the enhanced TCP. In Section 6 we study the behaviour of the enhanced TCP in various handoff scenarios.

6 Performance comparison: regular TCP vs. enhanced TCP

In this section we discuss the second set of experiments to evaluate the performance of enhanced TCP and compare its performance to regular TCP. The experimental setup used here is the same as in Section 3.

6.1 Handoff between same bandwidth, different delay links

Figure 17 Handoff from a low-delay link to a high-delay link with fixed bandwidth (1600 Kbps, 6400Kbps), (a) make-before-break handoff and (b) break-before-make handoff



Note: Transfer time for 100 packets after a make-before-break (a) and break-before-make handoff (b) between 6400 Kbps and 1600 Kbps links with varying delays of the old link. The new link delay is fixed at 300 ms. The disconnection period for break-before-make handoff is 500 ms.

We recall from Section 4.1 that the key problem affecting TCP performance in a make-before-break handoff from a low-delay to high-delay link is the occurrence of spurious RTOs along with the unnecessary congestion control actions associated with it.

Figure 17a shows the transfer time for 100 packets after a make-before-break handoff for 6400 Kbps links and 1600 Kbps links for regular TCP and enhanced TCP. The delay of the new link is fixed at 300 ms and the old link delay is varied as in Section 4.1. For the handoff between 6400 Kbps links enhanced TCP achieves a reduction in transfer time (median value) of 35–65% by avoiding spurious RTOs. When the ratio of the link delays is less than eight, enhanced TCP behaviour is similar to that of regular TCP as the spurious RTOs occur rarely in this situation. For handoff between 1600 Kbps links, enhanced TCP behaviour resembles the case of 6400 Kbps links described above and it achieves a reduction in transfer time up to 55%.

For low-bandwidth links such as 200 Kbps links, serialisation delay reduces the ratio between the old and new link delays which decreases the occurrence of spurious RTOs and enhanced TCP performance is only slightly better than that of regular TCP.

Figure 17b shows the transfer time for a break-before-make handoff for bandwidth values of 6400 Kbps and 1600 Kbps. With the disconnection period of 500 ms, the regular TCP will be in timeout recovery in most of the handoff points. Here the enhanced TCP applying algorithm A3 will retransmit the first unacknowledged packet immediately when the link comes up and reduce the *ssthresh* value only once thereby decreasing the delay in the recovery of lost packets allowing the TCP sender to continue with a larger window. In the case of 6400 Kbps links enhanced TCP reduces the transfer time up to 55%. The performance improvement of enhanced TCP is very significant (about 55%) for the handoffs from 75 ms delay link to 300 ms delay link. For a 1 ms delay link, the old link buffer is set to a minimum value of five packets and the link BDP is small. This allows only a slight performance improvement for enhanced TCP over regular TCP as *ssthresh* will have a low value anyway. For link bandwidth value of 1600 Kbps the

enhanced TCP reduces the transfer time up to 45 %. The algorithm A3 is invoked if TCP is in RTO recovery when the handoff notification arrives. For the handoff from 150 ms link, TCP will not be in RTO recovery and there is no performance improvement in using enhanced TCP in this case.

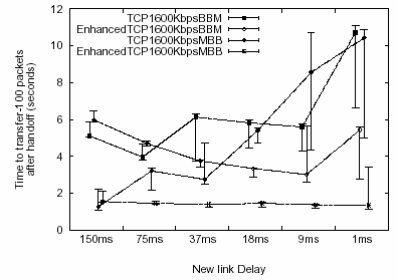
Figure 18 shows a comparison of the transfer time for regular TCP and enhanced TCP after make-before-break and break-before-make handoffs from a high-delay link to a low-delay link when both links have the same bandwidth of 1600 Kbps.

As discussed in Section 4.1 the main problems of the regular TCP in a make-before-break handoff from a high-delay link to a low-delay link (links of the same bandwidth) are:

1 decrease in the link BDP after a handoff resulting in congestion-related packet losses

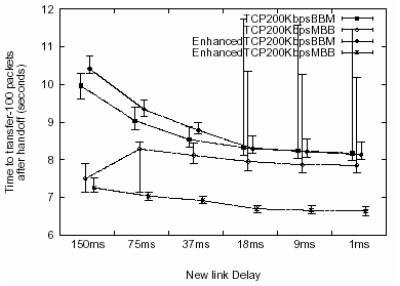
2 the slow convergence of RTO to the low delay of the new link.

Figure 18 Handoff from a high-delay link to a low-delay link with fixed bandwidth (1600 Kbps)



Note: Transfer time for 100 packets after a make-before-break (MBB) and break-before-make (BBM) handoff between 1600 Kbps links with varying new link delays and the delay of the old link is fixed at 300 ms**.**

Figure 19 Handoff from a high-delay link to a low-delay link with fixed bandwidth (200 Kbps)

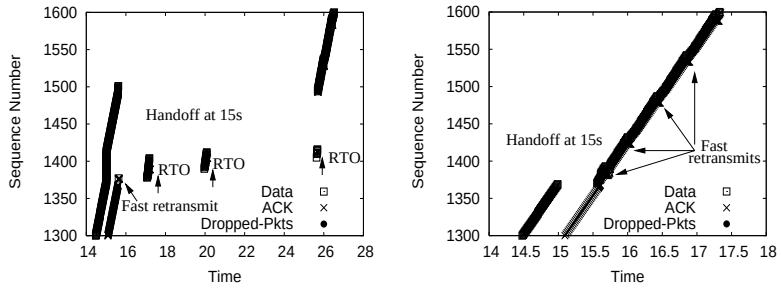


Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff between 200 Kbps links with varying new link delays and the delay of the old link is fixed at 300 ms.

Algorithms A2 and A4 of enhanced TCP enable it to mitigate these problems. Algorithm A2 enables enhanced TCP to minimise the packet losses by setting the *cwnd* and *ssthresh*

to the BDP of the new link. Algorithm A4 helps a rapid convergence of RTO to the RTT value of the new path. In Figure 18, we can see that there is up to 85 % reduction in the transfer time with enhanced TCP. Similar improvement can be observed with the handoff for 6400 Kbps links.

Figure 20 Comparison of regular TCP (a) and enhanced TCP (b): make-before-break handoff at 15.0s, between 1600Kbps links from 300ms to 1 ms



To understand the effect of a large decrease in BDP on regular TCP let us consider a specific example shown in Figure 20a. Here the handoff between 1600 Kbps links occurs at 15.0 seconds with the link delay changing from 300 ms to 1 ms, i.e., the link BDP decreases from 80 packets to one packet. There are 124 packets outstanding at the time of handoff. Many packets are lost due to the very low buffering capacity of the new link (approximately six packets). TCP fast retransmits the first unacknowledged packet at 15.62 seconds. TCP is not able to recover all the lost packets with the fast recovery and an RTO recovery occurs at 17.08 seconds. The new value of *ssthresh* (61 packets) is still significantly larger than the buffering capacity of the new link. The TCP sender increases *cwnd* in slow start relatively fast beyond the capacity of the new link, resulting in several packet losses and another RTO recovery is triggered at 20.01 seconds. One more RTO recovery at 25.77 seconds is necessary to recover all the lost packets yielding very poor performance. On the other hand, we can see from Figure 20b that, immediately after the handoff, the enhanced TCP stops injecting more segments to the network as *cwnd* is reduced down to the BDP of the new link. This effectively avoids any congestion-related losses after the handoff. Once enough cumulative ACKs have arrived, the TCP sender continues transmitting new segments in congestion avoidance, resulting in an ordinary steady-state behaviour with an occasional packet drop and subsequent fast retransmit and *cwnd* reduction.

For the break-before-make handoffs, we can see from Figure 18 that there is about 35-50% reduction in transfer time except in the case of handoffs from a 300 ms link to links with delays of 150 ms and 75 ms. RTO recovery is essential to recover the lost packets in all the scenarios. When the BDP decrease is at least a factor of eight, setting the *ssthresh* to half the *FlightSize* again results in losses in the RTO recovery phase leading to a further RTO. By contrast, the algorithm A2 enables enhanced TCP to set the *cwnd* and *ssthresh* to the BDP of the new link avoiding the losses during the RTO recovery. However, for the handoffs to a 150 and 75 ms delay links, setting the *ssthresh*

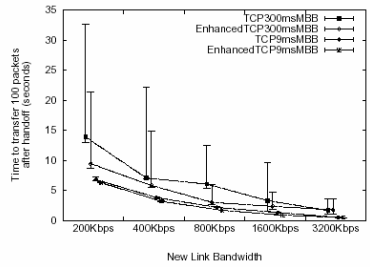
to BDP of the new link underutilises the path slightly resulting in about a 10% increase in the transfer time for enhanced TCP.

Figure 19 shows the transfer time for the make-before-break and break-before-make handoffs between 200 Kbps links. For the make-before-break handoffs the enhanced TCP reduces the transfer time by 15%. It is interesting to note that the upper quartile values of the transfer time for regular TCP is almost double that of the corresponding value for enhanced TCP. This high value is due to the reduction in sending rate by the occurrence of more than one RTO to recover the lost packets. The underutilisation of the link causes a slight increase in the transfer time for the enhanced TCP with break-before-make handoffs to moderately high delay links (150 ms, 75ms and 37 ms). Setting the *ssthresh* to the new link BDP is beneficial in handoffs to small delay links (18 ms, 9 ms and 1 ms). Enhanced TCP has a comparable median and upper quartile values whereas for regular TCP the upper quartile is about 25% larger than the median due to the occurrence of RTOs.

6.2 Handoff between same Delay, different BW Links

Figure 21 compares the time taken by regular TCP and enhanced TCP to transfer 100 packets after a make-before-break handoff for 300 ms and 9 ms delay links.

Figure 21 Handoff from a high bandwidth link to a low-bandwidth link with fixed delay (9 ms, 300 ms)



Note: Transfer time for 100 packets after a make-before-break (MBB) handoff between same delay links of 300 ms and 9 ms with varying new link bandwidth. The old link bandwidth is fixed at 6400 Kbps.

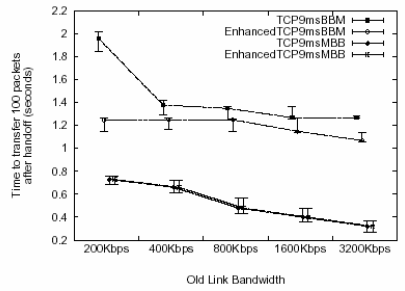
For the 300 ms delay link with regular TCP, a make-before-break handoff from a high-bandwidth to a low-bandwidth link results in congestion-related packet losses. The enhanced TCP by applying algorithm A2 sets *ssthresh* and *cwnd* to the BDP of the new link to avoid packet losses due to congestion and this reduces the transfer time (median) by about 25–35%. The Figure 21 also shows that there is at least a 25% reduction in the upper quartile values of the transfer time for enhanced TCP. In the case of handoff between links 9 ms delay the slight reduction in transfer time (5%) is due to algorithm A1 used in enhanced TCP to avoid spurious RTOs.

With the break-before-make handoff between 300 ms delay links, enhanced TCP behaves similarly to regular TCP as the retransmission timer will not expire during the disconnection period of 500 ms. In the case of a break-before-make handoff between 9 ms delay links, TCP will be in RTO recovery when the handoff notification arrives and

this causes enhanced TCP to invoke the algorithm A3 to immediately retransmit the first unacknowledged packet. As the BDP of the new link is small, the *cwnd* and *ssthresh* values are also small and restoring the *cwnd* and *ssthresh* values using the algorithm A3 does not bring much performance improvement to enhanced TCP.

Figure 22 compares the make-before-break and break-before-make handoff performances of regular and enhanced TCPs for the 9 ms delay links. The problem that TCP faces here is its inability to efficiently utilise the high-bandwidth available after a handoff. In the case of a make-before-break handoff from a low-bandwidth to a high-bandwidth link, there is no improvement in enhanced TCP performance as our algorithms are conservative in nature and do not attempt to blindly increase the *cwnd* in the absence of adequate information about the end-to-end path (Sarolahti et al., 2007b). It can be seen in Figure 22 that for break-before-make handoff between 9 ms links there is 10–40% reduction in transfer time with enhanced TCP due to the algorithm A3. In the case of handoff from a 200 Kbps link to a 6400 Kbps link, the unused connection time is about 700 ms for regular TCP whereas for enhanced TCP avoids this delay by applying algorithm A3 and reduces the transfer time by 40%. For higher bandwidths, the reduction in transfer time for the enhanced TCP is only 10% as the unused connection time has the much smaller value of 100 ms.

Figure 22 Handoff from a low-bandwidth link to a high-bandwidth link with fixed delay (9 ms)



Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff between 9 ms links with varying old link bandwidth. The new link bandwidth is fixed at 6400 Kbps.

6.3 Handoff between links of the same bandwidth-delay product (BDP) with different bandwidth and delay

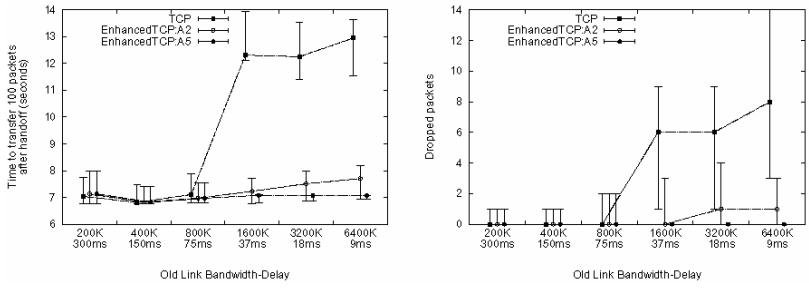
In this set of experiments we have two sets of fixed bandwidth/delay values namely, 200 Kbps/300 ms and 6400 Kbps/9 ms.

Figure 23a shows the time taken to transfer 100 packets after a make-before-break handoff from a high-bandwidth/low-delay link to a 200 Kbps/300 ms link. The problems arising in a make-before-break handoff are the occurrence of spurious RTOs and the packet losses due to the change in bandwidth. A couple of observations can be made here. Starting from the ratio of eight (between bandwidth and delay of the new link to that of the old link) enhanced TCP yields up to 45% reduction in transfer time (median). With the regular TCP spurious RTOs occur in almost all handoff points while enhanced

TCP avoids the spurious RTOs by using the algorithm A1 and reduces the packet losses by using the algorithm A2. Enhanced TCP behaves similarly to the regular TCP when the above ratio is less than eight.

We observe that there are scenarios where the algorithm A2 is not invoked though it could be beneficial to apply it. If there is a significant reduction in bandwidth due to a make-before-break handoff, segment burst in the new link can cause packet losses even if the new link capacity is not reached when the handoff completes. Figure 24a and Figure 24b present a comparison of regular TCP and enhanced TCP in one such scenario when a make-before-break handoff takes place from a 1600 Kbps/37 ms link to a 200 Kbps /300 ms link at 8.9 seconds. The BDP of both the links is ten packets. Due to the increase in link delay after the handoff spurious RTO occurs at 9.1 seconds and the regular TCP retransmits the first unacknowledged segment. The late ACKs for the segments sent before the handoff, start arriving at 9.2 seconds triggering unnecessary retransmissions. As the late ACKs arrive back-to-back roughly at the line rate of the old high-bandwidth link, unnecessary retransmissions are triggered at a rate which far exceeds the capacity of the new link. Therefore the new link is congested soon and we can see from Figure 24a that many retransmissions are lost. The late ACKs for data segments (from sequence number 1109 onwards) that took the new path after the handoff start arriving at 9.6 seconds triggering more new packets to be sent to the already congested new link. It is interesting to note that the new packets sent after the handoff are also dropped. As TCP is already in RTO recovery, the duplicate acknowledgements are not taken as an indication of a new instance of congestion (Allman et al., 1999) and regular TCP needs another RTO at 10.18 seconds to recover the lost packets. The recovery takes more time with the high-delay link and the performance of regular TCP is drastically affected.

Figure 23 Handoff between same BDP links (from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link), (a) transfer time and (b) dropped packets

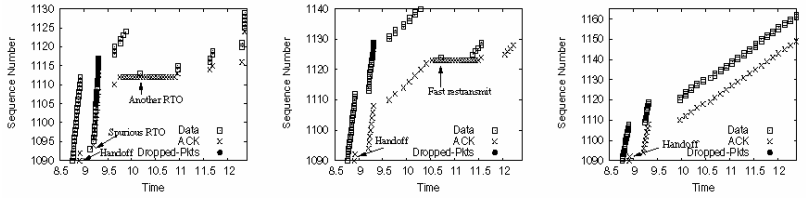


Note: Transfer time for 100 packets (a) and the number of dropped packets (a) after a make-before-break handoff between the same BDP links with varying bandwidth and delay of the old link and bandwidth and delay of the new link are fixed at 200Kbps/300ms.

We observe in Figure 24b that the enhanced TCP avoids spurious RTO in the same scenario but incurs the loss of the last new packets sent after the handoff even though these losses are fewer than the losses of regular TCP. The late ACKs for the segments sent before the handoff start arriving back-to-back at 9.2 seconds roughly at the line rate of the old high-bandwidth link triggering 16 new packets which congest the new link.

This is because the algorithm A2 is not at all invoked as the *FlightSize* is less than twice the BDP of the new link and so there is no consequent window reduction. A fast retransmit at 10.7 seconds is needed to recover these losses. The algorithm A5 given in Figure 25 is a modification of the algorithm A2 to make it effective when there is significant reduction in bandwidth due to a handoff. As we have noted earlier for make-before-break handoffs between same BDP links that the regular TCP performance becomes poor when the bandwidth and delay change by a factor of 8 or more. So in algorithm A5, if the bandwidth of the old link is at least 8 times the bandwidth of the new link, algorithm A5 sets the *cwnd* and *ssthresh* to the BDP of the new link when the *FlightSize* is greater than 1.5 times the BDP of the new link. Otherwise, as in algorithm A2, the *cwnd* and *ssthresh* is set to the BDP of the new link when the *FlightSize* is greater than two times the BDP of the new link.

Figure 24 Comparison of regular TCP (a), enhanced TCP with algorithm A2 (b), and enhanced TCP with algorithm A5 (c): make-before-break handoff at 8.9s from a 1600Kbps/37ms link to a 200Kbps/300ms link



(a) (b) (c)

Figure 24c shows the effectiveness of algorithm A5. The window is reduced to the BDP of the new link as the ratio of decrease in bandwidth is eight even though the *FlightSize* is smaller than twice the BDP of the link. We can see that there are no packet losses incurred with enhanced TCP when using the algorithm A5. Figure 23a shows the transfer time taken by the regular TCP and the two versions of the enhanced TCP. The slight reduction in transfer time for the enhanced TCP using the algorithm A5 is due to elimination of packet losses as shown in Figure 23.

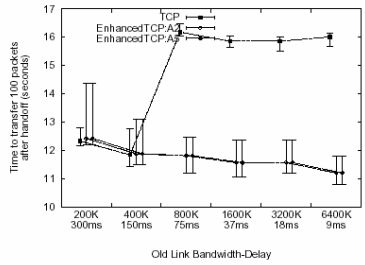
Figure 25 A5: Modified algorithm to reduce congestion related packet losses

```
When handoff notification arrives
 If ((TCP not in RTO recovery)
     If ( BW_oldlink >= 8 * BW_newlink)
        If (FlightSize > 1.5 * BDP_newlink)
            cwnd_reduction = 1
     Else if (FlightSize > 2 * BDP_newlink)
            cwnd_reduction = 1
      If (cwnd_reduction == 1)
       cwnd = max(2, BDP_newlink)
       ssthresh = cwnd
       cwnd_reduced = 1
```

Figure 26 shows the time taken to transfer 100 packets after a break-before-make handoff from varying bandwidth/delay links to a 200 kbps/300 ms link. We can see that immediate retransmission after a break-before-make handoff is beneficial with old link delay of 75 ms or less as in these cases the retransmission timer has already expired when the handoff notification arrives. In this case enhanced TCP retransmits the first unacknowledged packet immediately and restores the *ssthresh* value as in algorithm A2. Enhanced TCP behaves similarly to the regular TCP when the difference in the old and new link delays is small.

When the old link bandwidth/delay is fixed at 6400 Kbps/9 ms we observe that with a make-before-break handoff enhanced TCP performs better as it is able to avoid spurious RTOs with the increase in delay of the new link. In the break-before-make handoff, immediate retransmission improves the performance of enhanced TCP.

Figure 26 Handoff between same BDP links (from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link)



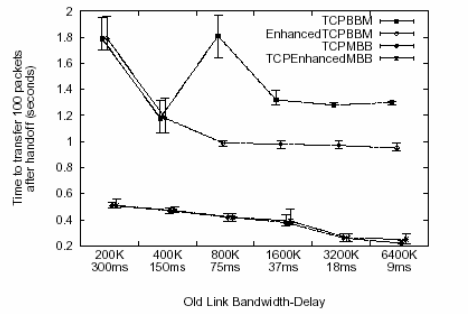
Note: Transfer time for 100 packets after a break-before-make handoff between the same BDP links with varying bandwidth and delay of the old link and the bandwidth and delay of the new link are fixed at 200Kbps/300ms. The disconnection period is 500 ms.

In one set of experiments we fix the bandwidth and delay of the old link to 200 Kbps /300 ms and the bandwidth and delay of the new link is varied. The main problems that TCP faces with a make-before-break handoff in this scenario are packet reordering and the inability to utilise the high bandwidth available after a handoff. Packet reordering in vertical handoff scenarios is a problem in its own right and we are not addressing it in the present study. Sarolahti et al. (2006) describes how Quick-Start can be used to effectively utilise the high bandwidth available after a handoff.

In the case of break-before-make handoff the retransmission timer is not likely to expire during the disconnection period of 500 ms and enhanced TCP behaves similarly to regular TCP. Figure 27 compares the make-before-break and break-before-make handoff performances of regular and enhanced TCPs when handoff occurs from a low-bandwidth/high-delay link to 6400 Kbps/9 ms link. We can see in Figure 27 that the regular TCP is unable to utilise the high bandwidth available after a make-before-break handoff. In break-before-make handoff with a disconnection period of 500 ms, TCP is in timeout recovery when the handoff notification arrives when the delay of the old link is less than 150 ms and enhanced TCP yields better performance by using algorithm A2. Figure 27 shows 20–45% reduction in transfer time with enhanced TCP when the delay of the old link is smaller than 150 ms. For the handoff from 300 ms delay link and

150 ms delay link, the retransmission timer may not expire during the disconnection period and enhanced TCP behaves similarly to regular TCP as the algorithm A3 will not be invoked.

Figure 27 Handoff between same BDP links (from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link)



Note: Transfer time for 100 packets after a make-before-break (MBB) and break-before-make (BBM) handoff for the same BDP links with varying bandwidth and delay of the old link and the bandwidth and delay of the new link are fixed at 6400 Kbps/9ms.

7 Conclusions and future works

In this paper we give a detailed study of the behaviour of TCP in the presence of vertical handoff and identify the problems of TCP in various vertical handoff scenarios. We then propose enhancements to the TCP sender algorithm to improve TCP performance in the presence of vertical handoffs. These algorithms are assisted with explicit cross-layer notifications indicating the changes in the access link characteristics. We group the changes due to the vertical handoff as arising from changes in delay, bandwidth and connectivity and propose enhancements to adapt TCP to various handoff scenarios. Our algorithms effectively address the problems arising from spurious RTOs, packet losses, prolonged disconnection and slow convergence to the new RTO value in a vertical handoff, yielding significant TCP performance improvement.

The study on the effect of vertical handoff on TCP and the algorithms proposed in this paper are based on our experiments with a single TCP flow. In the presence of multiple flows our proposed algorithms might not give expected performance improvements in all scenarios. Therefore, we would like to study how our proposed algorithms enable TCP to cope with a vertical handoff and tune these algorithms when necessary to get a better performance. We would like to study how to combine the link information in the explicit notifications with the information gathered through TCP probing the new network path and thereby trying to converge faster but safely to the new end-to-end RTT and available network capacity. Further it would be of interest to evaluate the algorithms in the setting of real networks.

References

Allman, M., Paxson, V. and Stevens, W. (1999) 'TCP congestion control', *RFC 2581*, April.

Blanton, E. and Allman, M. (2004) 'Using TCP duplicate selective acknowledgement (DSACKs) and stream control transmission protocol (SCTP) duplicate transmission sequence numbers (TSNs) to detect spurious retransmissions', *RFC 3708*, February.

Borman, D., Braden, R. and Jacobson, V. (1992) 'TCP extensions for high performance', *RFC 1323*, May.

Chakravorty, R., Vidales, P., Subramanian, K., Pratt, I. and Crowcroft, J. (2004) 'Performance issues with vertical handovers: experiences from GPRS cellular and WLAN hot-spots integration', in *Proc. IEEE Pervasive Communications and Computing Conference, (IEEE PerCom 2004)*, March, pp.155–164.

Daniel, L. and Kojo, M. (2006) 'Adapting TCP for vertical handoffs in wireless networks', in *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, November, pp.151–158.

El Malki, K. (ed.) (2007) 'Low latency handoffs in mobile IPv4', *RFC 4881*, June.

Gurtov, A. and Korhonen, J. (2004) 'Effect of vertical handovers on performance of TCP-friendly rate control', *ACM Mobile Computing and Communications Review*, July, Vol. 8, No. 3, pp.73–87.

Hansmann, W. and Frank, M. (2003) 'On things to happen during a TCP handover', in *Proc. 28th IEEE Conference on Local Computer Networks (LCN'03)*, October , pp.109–118.

Henderson, T. (ed.) (2007) 'End-host mobility and multihoming with the host identity protocol', internet-draft, March, work in progress.

Huang, H. and Cai, J. (2005) 'Improving TCP performance during soft vertical handoff', in *Proc. 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, March, Vol. 2, pp. 329–332.

Johnson, D., Perkins, C. and Arkko, J. (2004) 'Mobility support in IPv6', *RFC 3775*, June.

Keshav, S. (1991) 'A control-theoretic approach to flow control', in *Proceedings of ACM SIGCOMM*, pp.3–15.

Kim, S-E. and Copeland, J.A. (2003) 'TCP for seamless vertical handoff in hybrid mobile data networks', *In Proc. IEEE Globecom 2003. IEEE*, December.

Kim, S-E. and Copeland, J.A. (2004) 'Enhancing TCP performance for intersystem hand-off within heterogeneous mobile networks', in *Proceedings of IEEE Vehicular Technology Conference*, May.

Koodli, R. (ed.) (2005) 'Fast handovers for mobile IPv6', *RFC 4068*, July.

Lin, Y. and Chang, H. (2007) 'VA-TCP: a vertical handoff-aware TCP', in *Proceedings of the 2007 ACM symposium on Applied Computing*, ACM, pp.237–238.

Ludwig, R. and Meyer, M. (2003) 'The Eifel detection algorithm for TCP', *RFC 3522*, April.

Manner, J. and Kojo, M. (2004) 'Mobility related terminology', *RFC 3753*, June.

Matsushita, Y., Matsuda, T. and Yamamoto, M. (2007) 'TCP congestion control with ACK-pacing for vertical handover', *IEICE Transactions on Communications*, Vol. E90, No. B4, pp.885–893.

Moskowitz, R. and Nikander, P. (2006) 'Host identity protocol (HIP) architecture', *RFC 4423*, May.

Network Simulator ns-2. ns-2.29 (2005) Available at URL http://www.isi.edu/nsnam/ns.

Paxson, V. and Allman, M. (2000) 'Computing TCP's retransmission timer', *RFC 2988*, November.

Perkins, C. (ed.) (2002) 'IP mobility support for IPv4', *RFC 3344*, August.

Postel, J. (1981) 'Transmission control protocol', *RFC 793*, September.

Sarolahti, P. and Kojo, M. (2005) 'Forward RTO-recovery (F-RTO): an algorithm for detecting spurious retransmission timeouts with TCP and the stream control transmission protocol (SCTP)', *RFC 4138*, August.

Sarolahti, P., Kojo, M. and Raatikainen, K. (2003) 'F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts', *ACM SIGCOMM Computer Communication Review*, Vol. 33, No. 2, pp.51–63, April.

Sarolahti, P., Korhonen, J., Daniel, L. and Kojo, M. (2006) 'Using quick-start to improve TCP performance with vertical hand-offs', in *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, pp.897–904, November.

Sarolahti, P., Allman, M. and Floyd, S. (2007a) 'Determining an appropriate sending rate over an underutilized network path', *Computer Networks*, Elsevier, Vol. 51, No. 7, pp.1815–1832, May.

Sarolahti, P., Floyd, S. and Kojo, M. (2007b) 'Transport-layer considerations for explicit cross-layer indications', internet-draft, March, work in progress.

Savage, S., Cardwell, N., Wetherall, D. and Anderson, T. (1999) 'TCP congestion control with a misbehaving receiver', *Computer Communication Review*, Vol. 29, No. 5, pp.71–78, October.

Schütz, S., Eggert, L., Schmid, S. and Brunner, M. (2005) 'Protocol enhancements for intermittently connected hosts', *ACM Computer Communication Review*, Vol. 35, No. 2, pp.5–18, July.

Schütz, S., Koutsianas, N., Eggert, L., Eddy, W., Swami, Y. and Le, K. (2008) 'TCP response to lower-layer connectivity-change indications', internet-draft, February, work in progress.

Soliman, H., Castelluccia, C., El Malki, K. and Bellier, L. (2005) 'Hierarchical mobile IPv6 mobility management (HMIPv6)', *RFC 4140*, August.

Stemm, M. and Katz, R.H. (1998) 'Vertical handoffs in wireless overlay networks', *Mobile Networks and Applications*, Vol. 3, No. 4, pp.335–350.

Tsukamoto, K., Fukuda, Y., Hori, Y. and Oie, Y. (2006) 'New TCP congestion control scheme for multimodal mobile hosts', *IEICE Transactions on Communications*, Vol. E89-B, No. 6, pp.1825–1836.

Paper 3

III

Laila Daniel, Ilpo Järvinen and Markku Kojo

Combating Packet Reordering in Vertical Handoff using Cross-layer Notifications to TCP

In Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'08), October 2008, pages 297-303

IEEE International Conference on Wireless & Mobile Computing, Networking & Communication

Combating Packet Reordering in Vertical Handoff using Cross-layer Notifications to TCP

Laila Daniel, Ilpo Järvinen, Markku Kojo
Department of Computer Science
PL 68, University of Helsinki, 00014 Finland
Email: {firstname.secondname}@cs.helsinki.fi

Abstract

In this paper we propose an enhancement to the TCP sender algorithm to combat packet reordering that may occur due to a vertical handoff from a slow to a fast access link. The proposed algorithm employs cross-layer notifications regarding the changes in the access link characteristics. Our algorithm avoids unnecessary retransmissions by dynamically changing the dupthresh value according to the bandwidth and delay of the old and new access links involved in the handoff. In addition it uses DSACK information to infer that there are no congestion-related losses and selects proper values for cwnd and ssthresh after the handoff. Simulation results show that the unnecessary retransmissions caused by packet reordering in a vertical handoff can be effectively minimized over a wide range of bandwidth and delay ratios of the access links. In addition, our algorithm is effective in reducing the congestion-related packet losses due to a decrease in bandwidth-delay product (BDP) after a handoff.

1. Introduction

Packet reordering is not an uncommon phenomenon in the Internet [3]. The main reasons for the occurrence of packet reordering are local parallelism in high-speed routers, differentiated services, link layer retransmissions in wireless links and multi-path forwarding [22]. Multi-path forwarding may take place in some wireless overlay networks when a mobile node (MN) switches between different access technologies.

With the proliferation of wireless access technologies to the Internet, MNs equipped with multiple radio interfaces (for example, WLAN and GPRS/UMTS) are increasingly common. During the lifetime of a connection, an MN may switch among different access networks to have the best of connectivity, services quality, application needs and/or user preferences. Vertical handoff refers to the switching between the access points based on different link layer technologies [15]. The bandwidth and latency of the access links involved in a vertical handoff may differ by an order of magnitude. For example, in a handoff from GPRS to WLAN, the effective bandwidth typically increases from 200 Kbps to 5 Mbps (a maximum of 55 Mbps) while the latency decreases from 300 ms to less than 10 ms. In a make-before-break handoff [15], the connection to the old access router is broken only after the new connection is operational and for a while the packets can traverse both the paths. The differences in bandwidth and RTT of these paths may lead to packet reordering. In a break-before-make handoff packet reordering is not common as the connection to the old access router is broken before the connection to the new access router is made.

Packet reordering may have adverse effects on TCP. TCP relies on duplicate acknowledgements (*dupacks*) and retransmission timeouts (RTOs) to detect a packet loss. When a number of *dupacks* equal to a preset value (called *dupthresh*, usually 3) arrive at the TCP sender, TCP assumes that a packet loss has occurred, triggers the *fast retransmit* algorithm to retransmit the first unacknowledged segment, reduces the congestion window (*cwnd*) and slow-start threshold (*ssthresh*) and continues in *fast recovery* [2]. If the *dupacks* have been generated due to packet reordering, such *false fastretransmit* and the consequent reduction in sending rate degrades TCP performance [5, 22].

Several measures have been proposed to avoid the problem of packet reordering [5, 22, 4]. In all these schemes, *dupthresh* value is increased which enables TCP to avoid taking congestion control measures to a certain degree when reordering occurs but it increases the recovery time for dropped packets as timely action for packet losses cannot be taken. If TCP does not receive enough dupacks, e.g., due to small window size when a packet loss occurs, the TCP *fast retransmit* algorithm will not be invoked which results in a retransmission timeout that drastically reduces the throughput. The *nodupack* scheme proposed in [10] to combat the problem of packet reordering due to a vertical handoff suppresses the transmission of *dupacks* during a handoff and it may need timeout recovery in case of packet losses. So

978-0-7695-3393-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WiMob.2008.117

there is a tradeoff between timely detection of packet loss and making TCP robust to packet reordering.

Various techniques to increase the *dupthresh* values to avoid the triggering of the *fast retransmit* algorithm have been proposed in [5]. These techniques use a variant of the *Limited Transmit algorithm* [1] to preserve the ACK-clocking by sending new data for every second *dupack*. Reordering Robust-TCP (RR-TCP) proposed in [22] uses the DSACK [9] information to vary the *dupthresh* value adaptively for triggering the *fast retransmit* algorithm. It proposes several algorithms for avoiding the false retransmits proactively. TCP-NCR [4] roughly increase the *dupthresh* value based on the congestion window of data. TCP-NCR also extends TCP's *Limited Transmit algorithm* to allow the sending of new data during the period when the TCP sender is engaged in distinguishing between loss and reordering. TCP-NCR and TCP-RR try to avoid triggering false *fastretransmits* due to packet reordering, but these schemes do not take into account of the characteristics of the new link after a vertical handoff. This may lead to either underutilization of the new path or packet losses if the capacity of the new path is smaller. The algorithm we propose in this paper also tries to adapt to the characteristics of the new path after a vertical handoff.

Eifel [14] is designed to detect and avoid unnecessary retransmissions and also to undo congestion control measures already taken. If the bottleneck link bandwidth-delay product (BDP) of the new path after a vertical handoff is smaller than that of the old path, restoring the congestion window is likely to result in congestion on the new bottleneck link. According to [6], the congestion control measures that have been taken already should be undone only if all retransmitted packets in a particular window have been retransmitted unnecessarily. In a vertical handoff, as the path characteristics may change after the handoff, the TCP sender may have to wait for a very long time to confirm that all the retransmissions in a particular window were unnecessary. So in vertical handoff scenarios, to undo the congestion control measures already taken, the TCP sender needs additional information about the new path as there can be a significant change in the delay and the bandwidth of the paths involved in a handoff.

Estimating the changes in the end-to-end path properties after a vertical handoff is difficult as well as time consuming. If the TCP sender is explicitly notified about the changes in the access link properties such as bandwidth and delay due to a vertical handoff it can infer the possibility of packet reordering and defer from invoking the *fast retransmit* algorithm even when three dupacks arrive. The TCP layer on an MN can be locally informed of the changes in the attached access link characteristics by using a cross-layer notification. As the TCP layer at the other end of the connection, at the correspondent node (CN), is not aware of such changes, the introduction of an explicit end-to-end notification will help TCP to adapt to the changes due to a vertical handoff [20, 7, 19, 8, 12, 13].

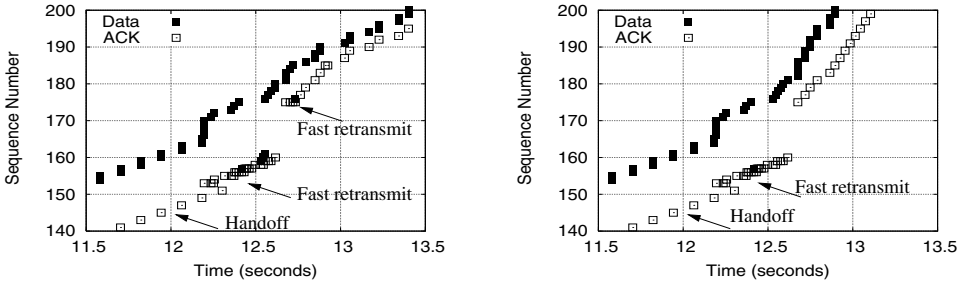
This paper reports a follow-up work on the research presented in our earlier papers [7, 8]. In those papers we described the various problems arising from a vertical handoff including packet reordering and proposed algorithms to mitigate the effect of some of these problems such as spurious RTOs, congestion-related packet losses, fast convergence to the new RTT and unused connection time. However, combating the effect of packet reordering was left for future work. In this paper we propose a solution to the problem of packet reordering due to a vertical handoff by introducing an enhancement to the TCP sender algorithm which makes use of the cross-layer notifications about the bandwidth and delay of the access links involved in a handoff. The resulting TCP adaptively determines a *dupthresh* based on the bandwidth and delay of the old and the new links. After a vertical handoff, it sets the *cwnd* and *ssthresh* based on the access link parameters and DSACK information. Experimental results show that our enhanced TCP algorithm is able to adapt to a vertical handoff by minimizing the unnecessary retransmissions.

The rest of the paper is organized as follows. In Section 2 we describe the problem of packet reordering arising from a vertical handoff. In Section 3 we present the algorithm to mitigate the effects of packet reordering. In Section 4 we evaluate the performance of the proposed TCP algorithm in various vertical handoff scenarios. In Section 5 we present the conclusions of this study.

2. Packet reordering due to vertical handoff

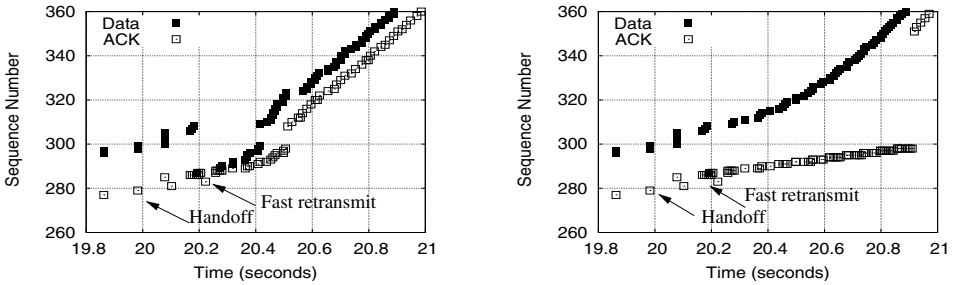
Packet reordering may occur when a make-before-break handoff occurs from a slow access link to a fast access link [10, 7]. During a make-before-break handoff, an MN can receive packets through the old link as well as through the new link. The sequence numbers of the packets arriving through the new link are greater than the expected sequence number as the packets with high sequence numbers sent after the handoff through the fast new link arrive at the TCP receiver earlier than the packets sent before the handoff through the slow old link. As a consequence of this reordering, the TCP receiver sends *dupacks* over the new link. When the TCP sender gets three *dupacks*, it triggers *fast retransmit* and *fast recovery* algorithms, and as a result the *cwnd* and the *ssthresh* are reduced. As the *dupacks* arrive not due to packet loss but due to reordering, the retransmissions are unnecessary. The *cwnd* reduction is undesirable unless the BDP of the new path is smaller than that of the old path.

We study the effect of reordering due to a vertical handoff using ns-2 [17] simulations. The baseline TCP for our



(a) Regular TCP (b) Enhanced TCP

Figure 1: Comparison of regular TCP and enhanced TCP: make-before-break handoff at 12 sec from a 200 Kbps/300 ms link to a 800 Kbps/75 ms link.



(a) Regular TCP (b) Enhanced TCP

Figure 2: Comparison of regular TCP and enhanced TCP: make-before-break handoff at 20 sec from a 200 Kbps/300 ms link to a 1600 Kbps/37 ms link.

study is the TCP-Sack1 algorithm in ns-2 and we refer to it as regular TCP. We now describe the behaviour of regular TCP for two packet reordering scenarios where the ratio of the delays (or bandwidths) of the old and the new links are such that the arrival pattern of *dupacks* differ. The BDP of the links involved in a handoff is kept constant in order to isolate the effect of reordering.

Consider a handoff from a 200 Kbps/300 ms link to a 800 Kbps/75 ms link, the ratio of the link delays is 4 and the two links have the same BDP of 10 packets. An interesting behaviour of regular TCP is shown in Figure 1a. A handoff occurs at 12.00 sec, and 8 out-of-order packets are received but they are not consecutive as the packets sent before handoff through the old link arrive at the receiver in-between. As the first two groups of *dupacks* consist of only two of them, the *fast retransmit* is triggered only by the third *dupacks* group at 12.40 sec. The last packet through the old link arrives earlier than the retransmissions through the new link and the fast recovery completes at 12.65 sec. The unnecessary retransmissions through the new link generate dupacks and TCP invokes *fast retransmit* unnecessarily again at 12.75 sec. Thus TCP reduces the *cwnd* twice which brings down the sending rate. While the reordering and TCP entering *fast retransmit* are observed to occur nearly at all handoff points, the behaviour described above where TCP subsequently goes into *fast retransmit* is observed in 20 % of the handoff points.

Figure 2a shows a time-sequence graph for a handoff from a 200 Kbps/300 ms link to a 1600 Kbps/37 ms link. In scenario of the Figure 2a the handoff occurs at 20.00 sec and the last packet sent before handoff arrives at the receiver at about 900 ms after the handoff. All forward progress of the flow happens over the new link starting from a reordered ACK at 20.07 sec. Then six packets are received out of order and their *dupacks* trigger fast retransmit at 20.19 sec. 12 packets are unnecessarily retransmitted.

We observe that as the ratio of the link delays increases, the *fast retransmit* algorithm is triggered immediately after a handoff and the number of unnecessary retransmissions increases. Our algorithm described in Section 3 minimizes these unnecessary retransmissions.

3. The proposed algorithm

As the end-to-end path characteristics of a TCP connection over a fixed Internet can be assumed to be relatively stable over the lifetime of the connection, we regard the changes in the path characteristics as arising from the changes in the access link characteristics due to a vertical handoff. We calculate a set of parameters from the access link characteristics. These parameters are taken as a rough estimate of the link characteristics and the algorithm makes a conservative use of these values in order to ensure that the lack of accuracy in determining these parameters does not make the algorithm aggressive.

Cross-layer notifications have been shown to be beneficial to TCP when path characteristics change widely due to a vertical handoff [20, 19, 7, 8, 21]. Many evaluations have been made on how to deliver the link characteristics information [20, 12, 13, 18, 21]. In essence, the TCP layer at the MN can be locally notified of the changes in the local link characteristics during a vertical handoff. This information can be sent to the TCP layer at the CN as a TCP option or along with the mobility registration message such as the binding update message in Mobile IPv6 [11] to be further forwarded to the TCP layer. A TCP sender can interpret the notification from the lower layers as a hint about the characteristics of the end-to-end path and adjust the congestion control parameters and RTO estimate so as to adapt to the new path in an efficient and timely manner. But even though a very conservative TCP approach is selected like with the *Response Connectivity Change Indication* (RLCI) mechanism [21] which forces TCP to slow-start if the new network path is unknown, unnecessary retransmissions may occur in the case of make-before-break handoffs.

In this work we model that the MN can deliver the handoff notification piggybacked in the mobility signalling messages so that it can be delivered to the TCP layer at the CN exactly when the handoff completes. Along with the cross-layer notification regarding the handoff, the TCP sender gets the information regarding the bandwidth and delay of the old and the new access links. The enhancements proposed here are TCP sender-specific and are invoked only upon the arrival of the handoff notification. So in the absence of the handoff notification, we get the performance of the regular TCP.

The enhancement to the TCP sender algorithm (with SACK [16] option enabled) to minimize the unnecessary retransmissions due to packet reordering in a vertical handoff is given in Figure 3. The parameters used in the proposed algorithm such as BDP of an access link and the RTT of the data segment-ACK pair traversing an access link are calculated as follows:

$BDP_{<link>} = BW_{<link>} \cdot RTT_{<link>}$

$RTT_{<link>} = 2 \cdot D_{<link>} + SDpkt_{<link>} + SDack_{<link>}$

where

$SDpkt_{<link>}$ - Data packet serialization delay on the access link

$SDack_{<link>}$ - Serialization delay of ACK on the access link

$D_{<link>}$ - Propagation delay of the access link

$BW_{<link>}$ - Bandwidth of the access link

When handoff notification arrives with the information regarding the old and the new access links

// Congestion possible due to bandwidth/BDP decrease?

If ($FlightSize > 2 \cdot BDP_{newlink}$)

Set $cwnd\_reduction$ to 1

If (($cwnd\_reduction = 1$) AND ($BDP_{oldlink} > BDP_{newlink}$) AND ($BW_{newlink} < 8 \cdot BW_{oldlink}$))

Set *cwnd* and *ssthresh* to $max(2, BDP_{newlink})$

If (TCP is not already in Loss recovery)

// False fast retransmit possible due to reordering?

If (($BW_{newlink} > 3 \cdot BW_{oldlink}$) AND ($cwnd\_reduction = 0$))

set $reordering\_flag$ to 1

dupthresh $= max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$

In Fast retransmit:

Retransmit the first unACKed segment

If ($reordering\_flag = 1$)

Save *cwnd* in $cwnd\_prev$

In Fast Recovery:

If ($reordering\_flag = 1$)

Send a new segment for every *dupack*

If (# of *dupacks* > *dupthresh*)

Set $return\_fastrecovery$ to 1

Return to the normal fast recovery

On the arrival of a new ACK indicating that all packets sent before handoff are ACKed:

Reset $cwnd\_reduction$

If ((DSACK indicates that the retransmission after the handoff was unnecessary) AND ($return\_fastrecovery = 0$) AND (*cwnd* $< min(cwnd\_prev, BDP_{newlink})$)

Set *cwnd* to min($cwnd\_prev, BDP_{newlink}$)

Set *ssthresh* to max($cwnd\_prev, BDP_{newlink}$)

Reset $reordering\_flag, return\_fastrecovery$

Reset *dupthresh* to 3

If (there is a significant change in delay)

Update the RTT variables

Figure 3: Algorithm to minimize the unnecessary retransmissions due to packet reordering in a vertical handoff

The enhanced TCP algorithm is invoked when a handoff notification arrives with the information regarding the old and the new access links. The algorithm is executed only if there is no imminent congestion in the new path. At the time of a handoff, if the *FlightSize* is greater than the buffering capacity of the new link, packet losses due to congestion may occur after the handoff. Therefore we check whether the *FlightSize* exceeds the estimated buffering capacity of the new link. We assume that the router queue size in front of the access link equals the BDP of the link so that the link has a total buffering capacity of twice the link BDP. If there is congestion, we set the flag *cwnd_reduction*, and modify *cwnd* and *ssthresh* as follows. The *cwnd* and *ssthresh* are set to $BDP_{newlink}$ if the BDP of the old link is greater than the BDP of the new link and the new link bandwidth is less than 8 times the old link bandwidth; otherwise *cwnd* and *ssthresh* remain unchanged. We observe from our experiments that if the new link bandwidth is greater than 8 times the old link bandwidth, the TCP sender is able to recover packet losses due to decrease in BDP using *fast recovery*. Reducing the *cwnd* in this scenario will reduce the sending rate thereby adversely affecting the performance of TCP. Any procedure related to DSACK detection and change of *dupthresh* is invoked only if there is no congestion in the new link at the time of handoff.

The enhanced algorithm makes use of the cross-layer information to determine whether reordering due to the handoff can lead to a false *fast retransmit*. If the ratio of the new bottleneck link bandwidth to the old bottleneck link bandwidth is greater than the *dupthresh*, enough *dupacks* may be generated to trigger a false *fast retransmit*. If this condition arises, the algorithm sets a flag called *reordering-flag* and $max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$ is set as the *dupthresh*. In *fast retransmit*, if the *reordering-flag* is set, TCP retransmits the first unacknowledged segment and saves the *cwnd* in a variable *cwnd_prev*. For every *dupack* which arrives in *fast recovery*, TCP sends a new segment to keep the ACK-clock running until one of the following events occurs: (1) If the total number of *dupacks* exceeds the *dupthresh*, TCP goes back to *fast recovery* and sets the flag *return_fastrecovery*, (2) If all the packets sent before handoff have been acknowledged, TCP resets the *dupthresh* back to the normal value of 3. If the segment retransmitted after the handoff is identified as unnecessary by the DSACK information, and TCP has not returned to *fast recovery*, the algorithm infers that the *dupacks* are generated by packet reordering and are not due to congestion and calculates the new *cwnd* and *ssthresh* values. If the current *cwnd* is less than both *cwnd_prev* and $BDP_{newlink}$, then TCP sets *cwnd* to the smaller of the *cwnd_prev* and the $BDP_{newlink}$ and *ssthresh* to the larger value so that TCP can slow start to find the new path characteristics.

The main advantages of our algorithm are (1) it is conservative in that it will not restore the *cwnd* and *ssthresh* if congestion exists in the network path, (2) it utilizes the new path partially while waiting for the packets to arrive through the old path, and (3) it effectively uses the DSACK information to set the *cwnd* and *ssthresh* of the new path.

4. Simulation results

In this set of experiments, we consider the vertical handoff scenarios where packet reordering occurs, i.e., the handoffs from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link. In order to study the effect of the change in BDP after a handoff, we categorize our experiments into the following three classes: (1) handoff between same BDP links, (2) handoff from a high BDP to a low BDP link, and (3) handoff from a low BDP to a high BDP link.

The simulation environment is the same as that described in our earlier paper [8]. We consider a single TCP flow from the CN to the MN. The TCP packet size is 1500 bytes with the TCP/IP headers included. In our experiments a 20-second interval is chosen to cover all the phases of a TCP connection and a handoff can occur uniformly in any of the 200 points at 100 ms intervals. The duration of each test run includes the completion of the handoff occurring in the 20-second interval. No link errors are modelled as we assume that the packet losses are due to congestion. This choice is justified as the present study is devoted to the effect of vertical handoff on TCP. In this set of experiments, the handoff occurs from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link. The old link bandwidth/delay is fixed at 200 Kbps/300 ms and the new link bandwidth/delay is varied.

To study the behaviour of TCP with vertical handoff we focus on how TCP behaves immediately after the handoff. As a performance index, we calculate the time taken to transfer (to get the acknowledgment) 'n' new data packets after a handoff through the new path where n varies from 50 to 200. We report the results only for the case where n is 100 as the results obtained are similar for all values of n.

4.1. Handoff between same BDP links

In this set of experiments the old link bandwidth/delay is fixed at 200 Kbps/300 ms and the new link bandwidth/delay is varied between 400 Kbps/150 ms and 6400 Kbps/9 ms as shown in Figure 4a. With regular TCP we observe that packet reordering due to a handoff from a high-delay link to a low-delay link triggers false *fast retransmits* in more than 80 % of the handoff points except in the case of a handoff to the 400 Kbps/150 ms link in which not enough *dupacks* are generated. As a result many packets are retransmitted

unnecessarily and *cwnd* and *ssthresh* are reduced. The corresponding results for enhanced TCP show that unnecessary retransmissions are avoided.

Figure 1 compares the behaviour of regular TCP and enhanced TCP for a handoff at 12 sec from 200 Kbps/300 ms to 800 Kbps/75 ms link. We can see that as enhanced TCP retransmits only the first unacknowledged segment, the fast retransmit triggered by the unnecessary retransmissions is avoided and enhanced TCP is able to send more packets than regular TCP which accounts for its slightly better (10 %) performance.

Figure 2 shows the behaviour of regular TCP and enhanced TCP when the handoff occurs at 20 sec from a 200 Kbps/300 ms link to a 1600 Kbps/37 ms link. Even though the time taken for both regular and enhanced TCP to send 100 packets is almost the same, enhanced TCP avoids unnecessary retransmissions and window reduction.

As the ratio of the delay of the old and new link increases (handoff to 6400 Kbps/9ms link) regular TCP is able to send more packets through the new fast link even though there are unnecessary retransmissions and window reduction. Enhanced TCP waits for all the packets through the old slow link to arrive and does not fully utilize the capacity of the new link even though it is transmitting one packet per *dupack* which results in poor performance (20 %). In such situations where the new link has a significantly higher bandwidth and lower delay than the old link, we have to examine whether it is preferable to wait for the packets through the old link so that we can avoid unnecessary retransmissions and *cwnd* reduction or to utilize the available capacity of the high-bandwidth link although a part of the capacity is wasted in unnecessary retransmissions.

4.2. Handoff from a high BDP link to a low BDP link

In this set of experiments the new link BDP is five packets half that of the old link. When there is a BDP decrease after a handoff our algorithm to mitigate the effect of packet reordering is not invoked as the fast retransmit algorithm may be triggered due to packet losses caused by BDP decrease. We can see in Figure 4b that in all handoff scenarios except the case of the handoff to a 400 Kbps/75 ms link enhanced TCP behaves similar to regular TCP. In the case of a handoff to a 400 Kbps/75 ms link, regular TCP needs RTO recovery to recover the lost packets due to BDP change whereas with enhanced TCP the packet losses are kept to a minimum as we set the *cwnd* to the BDP of the new link, yielding about 20 % reduction in transfer time. For the handoff to 800 Kbps/37 ms link, setting the *cwnd* to the BDP of the new link, reduces the packet losses but the reduction in sending rate nullifies the improvement achieved. For the handoffs to 1600Kbps/ 18 ms link, 3200Kbps/9 ms link and 6400 Kbps/ 4 ms link as our algorithm is not at all invoked the behaviour is similar to regular TCP.

4.3. Handoff from a low BDP link to a high BDP link

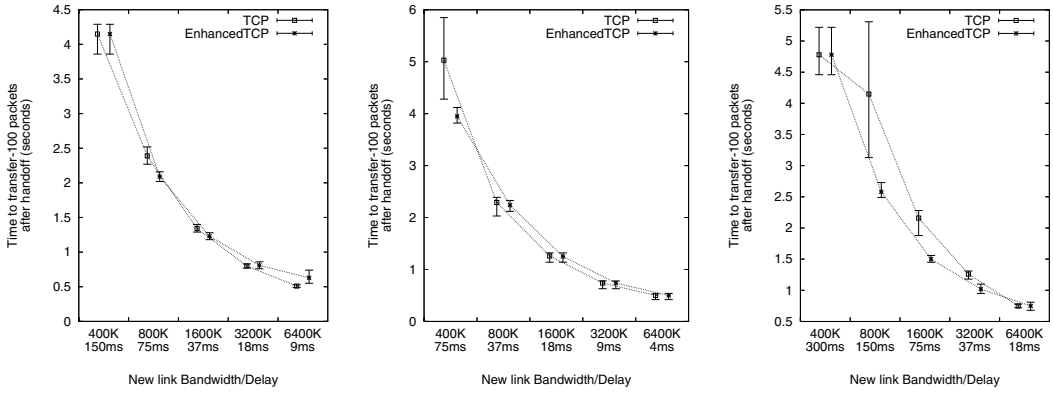
In this set of experiments the new link BDP is 20 packets, double that of the old link. As can be seen in the Figure 4c, the proposed algorithm is effective (up to 30 % reduction in transfer time) with the increase in BDP after a handoff. Our algorithm is not invoked in the handoff to 400 Kbps/ 300 ms link as there is no scenario leading to a false *fast retransmit*. In the case of handoff to 800Kbps/ 150 ms link, regular TCP suffers from multiple fast retransmits similar to the case described in Figure 1a resulting in high variation in the quartile values, whereas enhanced TCP yields a 38 % reduction in transfer time. For the cases of handoff to 3200 Kbps/37 ms and 6400 Kbps/ 18 ms links, enhanced TCP avoids unnecessary retransmissions even though the transfer time is comparable with that of regular TCP. In general, avoiding the unnecessary retransmissions as well as window reduction and slow starting to the new BDP makes the enhanced TCP perform better than regular TCP.

5. Conclusions and future work

In this paper we have proposed an enhancement to the TCP sender algorithm which makes use of the cross-layer notifications to avoid the problems of TCP due to packet reordering in a make-before-break vertical handoff. Simulation results presented in the paper show the effectiveness of the algorithm when the bandwidth(or delay) ratio of the access links varies over a wide range. Further study is required to see when it is preferable to utilize the high capacity link available after a handoff instead of taking measures to avoid unnecessary retransmissions.

As it is difficult as well as unreliable to obtain the exact bandwidth and delay of the access links, we have used the link characteristics available from the MN as hints or bounds for setting the initial values of the TCP congestion control parameters. We need to study how the information obtained by probing the end-to-end path characteristics can be integrated with the cross-layer notification for a faster convergence of TCP parameters after a vertical handoff.

The study on the effect of vertical handoff on TCP and the algorithm proposed are based on our experiments with a single TCP flow. In the presence of multiple flows, the algorithms are likely to require further enhancements. In addition to the experimentation of the algorithm with multiple flows we intend to evaluate the algorithm in real network environments.



(a) To same BDP (b) To low BDP (c) To high BDP

Figure 4: Transfer time for 100 packets after a make-before-break handoff (a) to a same BDP link (b) to a low BDP link (c) to a high BDP link with varying bandwidth/delay of the new link while the old link is fixed at 200Kbps/300ms (200 repetitions). The x-axis shows the link parameters and the y-axis shows the lower quartile, median, and upper quartile of the time (in seconds) to transfer 100 new packets after the handoff.

References

[1] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, Jan. 2001.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, Apr. 1999.

[3] J. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6), 1999.

[4] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton. Improving the Robustness of TCP to Non-Congestion Events. IETF RFC 4653, Apr. 2006.

[5] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM Computer Communication Review*, 32(1), Jan. 2002.

[6] E. Blanton and M. Allman. Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions. RFC 3708, Feb. 2004.

[7] L. Daniel and M. Kojo. Adapting TCP for Vertical Handoffs in Wireless Networks. In *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, Nov. 2006.

[8] L. Daniel and M. Kojo. Using Cross-layer Information to Improve TCP performance with Vertical Handoffs. In *Proc. 2nd International Conference on Access Networks (Accessnets 2007)*, Aug. 2007.

[9] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgment (SACK) Option for TCP. RFC 2883, July 2000.

[10] W. Hansmann and M. Frank. On Things to Happen During a TCP Handover. In *Proc. 28th IEEE Conference on Local Computer Networks (LCN'03)*, Oct. 2003.

[11] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

[12] J. Korhonen, A. Makela, S. Park, and H. Tschofenig. Link and Path Characteristic Information Delivery Analysis. Internet-Draft, Oct. 2006. Work in progress.

[13] J. Korhonen, S. Park, J. Zhang, C. Hwang, and P. Sarolahti. Link Characteristic Information for IP Mobility Problem Statement. Internet-Draft, June 2006. Work in progress.

[14] R. Ludwig and M. Meyer. The Eifel Detection Algorithm for TCP. RFC 3522, Apr. 2003.

[15] J. Manner and M. Kojo. Mobility Related Terminology. RFC 3753, June 2004.

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, Oct. 1996.

[17] Network Simulator ns-2. ns-2.29, 2005.

[18] S. Park, M. Lee, J. Korhonen, and Z. Zhang. Link Characteristic Information for Mobile IP. Internet-Draft, June 2005. Work in progress.

[19] P. Sarolahti, J. Korhonen, L. Daniel, and M. Kojo. Using Quick-Start to Improve TCP Performance with Vertical Hand-offs. In *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, Nov. 2006.

[20] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol Enhancements for Intermittently Connected Hosts. *ACM Computer Communication Review*, 35(2), July 2005.

[21] S. Schütz, N. Koutsianas, L. Eggert, W. Eddy, Y. Swami, and K. Le. TCP Response to Lower-Layer Connectivity-Change Indications. Internet-Draft, Feb. 2008. Work in progress.

[22] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A Reordering-Robust TCP with DSACK. In *Proc. 11th IEEE International Conference on Networking Protocols (ICNP 2003)*, Nov. 2003.

Paper 4

Laila Daniel and Markku Kojo

The Performance of Multiple TCP Flows with Vertical Handoff

In Proceedings of the 7th ACM International Symposium on Mobility Management and Wireless Access (MobiWac'09), October 2009, pages 17-25

IV

The Performance of Multiple TCP Flows with Vertical Handoff

Laila Daniel
Department of Computer Science
P.O. Box 68, University of Helsinki
FI-00014 Finland
laila.daniel@cs.helsinki.fi

Markku Kojo
Department of Computer Science
P.O. Box 68, University of Helsinki
FI-00014 Finland
markku.kojo@cs.helsinki.fi

ABSTRACT

The performance of an individual TCP flow with a vertical handoff has been studied in several papers. However, the effect of a vertical handoff on *multiple* TCP flows has been little studied. In this paper we study the behaviour of multiple competing TCP flows with a vertical handoff. As a part of this study we evaluate the cross-layer assisted TCP enhancements for a vertical handoff that we had earlier proposed and analyzed for a single TCP flow. We show that our algorithms can be adapted for multiple TCP flows with minor modifications and that they are effective in improving multiple flow-TCP performance in the presence of a vertical handoff.

Categories and Subject Descriptors

C.2.1 [**N**]: etwork Architecture and Design; C.2.2 [**N**]: etwork Protocols; C.4 [**P**]: erformance of Systems

General Terms

Performance, Algorithms

Keywords

Vertical Handoff, TCP, Wireless Access Networks, Cross-layer notifications

1. INTRODUCTION

The problem of Transmission Control Protocol (TCP) [25] behaviour in the presence of vertical handoffs [22] has grown in significance with the proliferation of wireless access networks to connect to the Internet and has been an active research area in recent years [8, 9, 12–15, 26, 31]. These studies have shown that the potentially significant differences in the bandwidth and/or delay of the two access links involved in a vertical handoff can affect TCP performance. The major problems of TCP due to a vertical handoff are the unnecessary retransmissions and congestion window (*cwnd*) reduction due to spurious RTOs and packet reordering as well as packet losses due to abrupt changes in the link capacity and link disconnection. Several cross-layer assisted enhancements have been proposed in the literature to mitigate these problems and to improve TCP performance [7–9, 12–15, 18–20, 26, 29, 30, 32]. These studies have focused on the effect of handoff on a *single* TCP flow. However, the effect of competing TCP flows has not been taken into account in evaluating the various proposed solutions. As multiple parallel TCP flows tend to affect TCP behaviour in general and more so in a vertical handoff, the resulting change in TCP behaviour has to be studied and taken into account in the proposed solutions. Such an approach in evaluating the adaptations of TCP algorithms in a dynamic environment has been suggested previously in [2] as a step closer to real-world networks.

In this paper we first study how TCP performance is affected by a vertical handoff when multiple TCP flows are present and then describe how the cross-layer assisted TCP enhancements for vertical handoff that we had proposed earlier for a single TCP flow [6,9] can be easily adapted for the case of multiple TCP flows. We show that with the inclusion of the number of simultaneous TCP flows in the cross-layer information, we can easily modify our earlier algorithms to adapt to multiple flow scenarios.

The rest of the paper is organized as follows. Section 2 gives a overview of the related research work on methods to improve TCP performance with a vertical handoff. Section 3 describes our algorithms to adapt TCP to a vertical handoff when multiple TCP flows are present. Section 4 describes the results of the simulation experiments to evaluate these algorithms. Section 5 presents the conclusions of the study.

2. RELATED WORK

We present an overview of the research dealing with the problems of TCP in the presence of vertical handoffs. The solutions proposed in these papers are in the context of a single TCP flow. To the best of our knowledge the behaviour of multiple TCP flows in a vertical handoff has not been described explicitly in the research literature. We categorize the related work based on the problems of vertical handoff they try to solve.

To avoid the problem of spurious retransmission timeouts (RTOs) that may occur due to a vertical handoff, [19] suggests that upon receiving a handoff notification, a TCP receiver should send the acknowledgement for a received packet through both the old and new interfaces and also reset the RTO value to 3 seconds. The use of ICMP messages to calculate the new RTO value after a handoff has been proposed in [26] as a solution to the spurious RTO problem. The TCP-Eifel detection algorithm [21] uses the TCP timestamps option [5] to detect spurious RTOs. Forward-Retransmission Timeout (F-RTO) algorithm [28] is a TCP sender-only algorithm that helps to detect spurious RTOs.

Setting the *cwnd* appropriately after a handoff is crucial both in avoiding the congestion-related losses due to a handoff to a lower

MobiWac'09, October 26–27, 2009, Tenerife, Canary Islands, Spain.

bandwidth-delay product (BDP) link as well as in effectively utilizing the higher BDP of a new link after a handoff. Slow-starting to find the new *cwnd* value after a handoff is proposed in [18]. The use of the Quick-Start algorithm [27] to find the correct sending rate with the help of routers is proposed in [29]. In the proposals [20, 32] the BDP of the end-to-end path after a handoff is calculated as the product of the bottleneck link capacity estimated using the packet-pair algorithm [17] and the round-trip time (RTT) and then the *cwnd* and slow-start threshold (*ssthresh*) are set to this BDP value. An explicit *cwnd* reduction trigger from the mobile node which signals the difference between the BDP values of the old and new link is proposed in [14]. *BDP probing*, proposed in [19], is a technique which initially sends two back-to-back packets just after a handoff to estimate the capacity of the bottleneck link and then sends the data packets at the rate of the estimated capacity to find the available bandwidth. A receiver-based mechanism proposed in [12] addresses the problem of abrupt change in link capacity due to a vertical handoff. When an impending handoff is detected, the TCP receiver sends the receiver advertised window (*rwnd*) based on the BDP of the new network and this *rwnd* will be effective once the mobility registration is complete. Overbuffering is suggested in [13] to reduce the effects of BDP change due to a handoff.

To mitigate the problems arising from packet reordering due to a vertical handoff, a *nodupack* scheme is proposed in [14] and it suppresses the transmission of *dupacks* during a handoff. However, this may delay the loss recovery if the reordering occurs due to a handoff. An RTT-equalization scheme is proposed in [26] which sends the acknowledgements for the packets received from the fast interface through the slow interface and vice versa. DSACK [11], an extension of SACK in which the receiver reports to the sender the receipt of a duplicate segment, can be used to detect packet reordering and to undo the consequent congestion control actions [3].

To minimize the unused connection time after a handoff, a TCP retransmission trigger that causes TCP to attempt a retransmission when the connectivity is restored after a handoff is proposed in [30].

A detailed account of the research work in this area can be found in Chapter 5 of [6].

3. TCP ENHANCEMENTS FOR MULTIPLE FLOWS

In this section we present a brief overview of the problems of TCP in vertical handoffs and describe the modifications we propose in this paper to our earlier algorithms [6,7,9] in order to adapt them for handoffs in the context of multiple TCP flows. Our algorithms make use of the cross-layer information regarding the occurrence of a handoff, the bandwidth and delay of the access links involved in the handoff and the number of TCP flows undergoing the handoff. Our modelling assumption is that the mobile node can send the above information to the TCP sender at the correspondent node along with the mobility signalling.

3.1 Spurious RTOs

Spurious RTOs occur when a make-before-break handoff [22] occurs from a low-latency link to a high-latency link. After the handoff, the acknowledgments (ACKs) will be delayed due to the higher delay of the new link. TCP retransmission timer expires before the arrival of the ACKs through the new link due to the small RTO value calculated on the basis of the old path. This spurious RTO will cause unnecessary retransmission of packets and reduction in *cwnd* and *ssthresh*. Unnecessary retransmissions waste the bandwidth and reduction in *cwnd* and *ssthresh* reduce the sending rate, resulting in performance degradation.

In order to avoid the occurrence of spurious RTOs we calculate the minimum RTO (*minrto*) based on the new access link delay and update the RTO timer immediately so that the new *minrto* comes into effect. As a result any change in the delay of the end-to-end path will be reflected better in the RTO calculation. As the *minrto* calculation is based on the access link delay alone, the RTT variables are initialized as in RFC 2988 upon the arrival of the ACK for the data sent through the new access link. This enables the RTO to adapt to the end-to-end RTT quickly. No modification to this algorithm is needed for multiple TCP flow scenarios. Due to space limitations this algorithm is not given here and the reader is referred to [6, 9] for details.

3.2 Packet Losses due to Congestion

Packet losses can occur when there is a decrease in BDP after a handoff. When a handoff occurs from a high-bandwidth link to a low-bandwidth link, significant packet losses can occur even when the BDP of the two links remains the same.

We first introduce the algorithm used to avoid the packet losses due to a decrease in BDP in the case of a single TCP flow and then describe the modification to this algorithm for the case of multiple TCP flows which is shown in Figure 1. We set the *cwnd* and *ssthresh* to the BDP of the new access link if the *FlightSize* at the time of handoff is greater than twice the BDP of the new access link (or 1.5 times the BDP of the new access link in case the bandwidths of the old and new link differ significantly). A detailed discussion of the rationale of this algorithm can be found in [6,9]. When there are multiple flows sharing a bottleneck access link it is no longer appropriate to set the *cwnd* of each TCP flow to the BDP of the new link. Accordingly the *cwnd* and *ssthresh*) is set to the BDP of the new link divided by the number of concurrent flows that use the link at the time of handoff. This value corresponds to a single flow's share of the bandwidth when a number of flows share the bottleneck link. Here we make the assumption is that the bottleneck link is the last/first hop wireless access link. This assumption is justifiable as the bandwidth of the wired links in an end-to-end path is usually much higher than that of the wireless access links at its end points.

When a handoff notification arrives
If ((TCP not in RTO recovery)
If ($BW_{oldlink} >= 8 * BW_{newlink}$)
If ($\mathbf{FlightSize > 1.5 * (BDP_{newlink}/N)}$)
/* N refers to the number of flows */
$cwnd\_reduction = 1$
Else if ($BW_{oldlink} < 8 * BW_{newlink}$)
If ($\mathbf{FlightSize > 2 * (BDP_{newlink}/N)}$)
$cwnd\_reduction = 1$
If ($cwnd\_reduction == 1$)
$\mathbf{cwnd = max(2, BDP_{newlink}/N)}$
$ssthresh = cwnd$
$cwnd\_reduced = 1$

Figure 1: Algorithm to reduce congestion-related packet losses for multiple flows. Adaptation to multiple flows shown in bold.

Figures 2 and 3 describe our algorithms to reduce the unused connection time and to combat the problems arising from packet reordering in a vertical handoff in the setting of multiple TCP flows. In these algorithms also we set the *cwnd* and *ssthresh* in the same manner based on the number of simultaneous TCP flows through

the bottleneck link. This is the only modification necessary to adapt our algorithms designed for a single TCP flow to the case of multiple TCP flows in the presence of a vertical handoff. As the information about the number of flows can be easily included in the handoff notifications, our algorithms are easy to implement in practice.

3.3 Unused Connection Time and ssthresh Reduction

When a handoff notification arrives:
If (TCP in RTO recovery)
Retransmit the first unacknowledged packet
Set *ssthresh* to $\max(\mathbf{2}, \mathbf{BDP_{newlink}/N})$
/* N refers to the number of flows */
If there is a significant change in delay
Initialize RTT variables as for a new connection
When ACK for new data arrives
Update RTT variables

Figure 2: Algorithm to reduce the unused connection time and to set *ssthresh*. Adaptation to multiple flows shown in bold.

Figure 2 gives our algorithm to reduce the unused connection time and to set the *ssthresh* in a break-before-make handoff. When a break-before-make handoff occurs, the end-to-end path between the mobile node and correspondent node is broken and the connectivity resumes only after the handoff is completed resulting in packet losses. If the disconnection period is greater than the current RTO value, the retransmission timer expires. The TCP sender retransmits the first unacknowledged segment and doubles the RTO value. For each subsequent timer expiration for the same segment, TCP doubles the RTO value again [24]. When the end-to-end connection is up, the TCP sender needs to wait until the retransmission timer expires again before attempting another retransmission. This unused connection time can be up to one minute [24] depending on the disconnection length and the next scheduled RTO and it increases the recovery time of the lost packets. If more than one timeout has occurred, i.e., a retransmission is considered to be lost, the *ssthresh* value is further reduced in some implementations resulting in inefficient recovery. By retransmitting the first unacknowledged segment immediately if the TCP sender is in RTO recovery when the handoff notification arrives and setting the *ssthresh* to the BDP of the new link scaled by the number of flows, our algorithm given in Figure 2 is able to mitigate the problems due to a long disconnection in a break-before-make handoff.

3.4 Packet Reordering

Packet reordering is a problem that TCP faces when there is a significant reduction in delay after a make-before-break handoff. The packets sent through a low-delay link after the handoff may overtake the packets transmitted through the high-delay link before the handoff and this causes packet reordering which generates *dupacks*. If the TCP sender receives a *dupthresh* number of *dupacks* (typically 3) it enters fast recovery, halves the *ssthresh* and *cwnd* and continues in congestion avoidance. The retransmission and the reduction in *ssthresh* and *cwnd* are unnecessary as the *dupacks* which arrive are due to packet reordering and not due to congestion.

We briefly describe the main idea behind the algorithm given in Figure 3 and refer the reader to [7] for its detailed description. If the bandwidth of the new access link is greater than *dupthresh* times the bandwidth of the old access link, there is a possibility of packet reordering leading to false fast retransmit. If this condition arises, a new *dupthresh* value is calculated based on the ratio of the bandwidth of the two access links. In fast retransmit, the TCP sender saves the previous *cwnd* value if there is a possibility of reordering. In fast recovery, the TCP sender sends a new segment for every arriving *dupack* until all the segments transmitted before the handoff are acknowledged or the number of *dupacks* exceeds the *dupthresh*. In the latter case TCP returns to the normal fast recovery [10]. If the retransmission is identified as unnecessary using DSACK information, the *cwnd* and *ssthresh* are set to the BDP of the new access link scaled by the number of flows.

When a handoff notification arrives with the information regarding the old and the new access links
/* Congestion likely due to bandwidth or BDP decrease ? */
If ($\mathbf{FlightSize} > \mathbf{2} \cdot (\mathbf{BDP_{newlink}/N})$)
Set $cwnd\_reduction$ to 1
If (($cwnd\_reduction = 1$) AND
($BDP_{oldlink} > BDP_{newlink}$) AND
($BW_{newlink} < 8 \cdot BW_{oldlink}$))
Set *cwnd* and *ssthresh* to $\max(\mathbf{2}, (\mathbf{BDP_{newlink}/N}))$
/* N refers to the number of flows */
If (TCP is not already in Loss recovery)
/* False fast retransmit likely due to reordering ? */
If (($BW_{newlink} > 3 \cdot BW_{oldlink}$) AND
($cwnd\_reduction = 0$))
set $reordering\_flag$ to 1
dupthresh $= max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$
In Fast retransmit:
Retransmit the first unACKed segment
If ($reordering\_flag = 1$)
Save *cwnd* in $cwnd\_prev$
In Fast Recovery:
If ($reordering\_flag = 1$)
Send a new segment for every *dupack*
If (number of *dupacks* $>$ *dupthresh*)
Set $return\_fastrecovery$ to 1
Return to the normal fast recovery
On the arrival of a new ACK indicating that all packets sent before handoff are ACKed:
Reset $cwnd\_reduction$
If ((DSACK indicates that the retransmission after the handoff was unnecessary) AND
($return\_fastrecovery = 0$) **AND**
($\mathbf{cwnd} < \mathbf{min}(\mathbf{cwnd\_prev}, (\mathbf{BDP_{newlink}/N})$)
Set *cwnd* to min(cwnd_prev, $(\mathbf{BDP_{newlink}/N})$
Set *ssthresh* to max(cwnd_prev, $(\mathbf{BDP_{newlink}/N})$
Reset $reordering\_flag, return\_fastrecovery$
Reset *dupthresh* to 3
If (there is a significant change in delay)
Update the RTT variables

Figure 3: Algorithm to combat the problems arising from packet reordering in a vertical handoff. Adaptation to multiple flows shown in bold.

3.5 Slow RTO convergence

After a handoff, the RTO will converge to the RTT of the new path very slowly. One reason for this is that the formula for updating the smoothed RTT (SRTT) value at the TCP sender gives a much smaller weight to the current RTT sample compared to that of the previous SRTT value. Another reason is that the RTT variables are updated only once in an RTT and not for each ACK received [24]. In order to quickly converge to the RTO value corre-

sponding to the new path, we initialize the RTT variables as in RFC 2988 if the old and the new RTT values differ by a factor of two or more and the new RTT value is greater than *minrto*. No modification to this algorithm is needed for multiple TCP flow scenarios. Due to space limitations we do not give this algorithm here and the reader refer to to [6, 9].

4. SIMULATION RESULTS

We use the ns-2 network simulator to model the behaviour of TCP in a vertical handoff. This section describes the simulation experiments and our results obtained. We first discuss the simulation setup, and then move on to investigating different types of vertical handoff scenarios.

4.1 Simulation Setup

In the simulation model the mobile node is capable of using both the access links involved in a vertical handoff. Both access links have dedicated base stations that are connected to a common wireless access router by 100 Mbps links. The router has a 100 Mbps connection to a server (correspondent node) in the fixed network. The propagation delay over each of the fixed links is 2 ms. Unless otherwise stated, the router buffer size of each link is set to $max(BDP_{<link>}, 5)$ packets. We consider bulk TCP flows from the correspondent node to the mobile node. A detailed description of the simulation environment is given in [6, 9].

The baseline TCP used in the experiments is TCP SACK [4] and is referred to as regular TCP. TCP SACK with the enhancements we had proposed earlier [6, 9] is referred to as Enhanced-TCPv0 (ETCPv0) and the TCP SACK in conjunction with the algorithms described in Section 3 is referred to as Enhanced-TCPv1 (ETCPv1). The TCP packet size is 1500 bytes including the TCP/IP headers.

In our experiments a 20-second interval is chosen to cover all the phases of a TCP connection and a handoff can occur uniformly in any one of the 200 instances at 100 ms intervals. The duration of each test run includes the completion of the handoff occurring in the 20-second interval. No link errors are modelled as we assume that the packet losses are solely due to congestion.

In order to study the behaviour of TCP with vertical handoff we focus on the TCP behaviour immediately after a handoff. As a performance metric, we calculate the time taken to transfer (to receive the acknowledgment) a specific number of packets. This time is calculated with respect to the slowest flow. In comparing the performance of the enhanced TCP with the regular TCP, we use the median value of the time taken to transfer 100 packets after a handoff. In all the performance graphs given in this paper, the x-axis shows the number of flows and the y-axis shows the lower quartile, median, and upper quartile of the time (in seconds) to transfer 100 packets after the handoff.

We are categorizing our experiments into two classes, (i) handoff from a fast link to a slow link and (ii) handoff from a slow link to a fast link. We have chosen the following four sets of bandwidth and delay combinations to reflect the situations arising in handoff involving access networks such as EGPRS [23], HSDPA [1], WiMAX [33] and WLAN [16]. A rough range of the bandwidth and propagation delay (one-way) of the access networks such as EGPRS (200 Kbps/300 ms), HSDPA (700 Kbps/75 ms, 2000 Kbps/50 ms, 6000 Kbps/50 ms), WiMax (2000 Kbps/50 ms, 11000 Kbps/20 ms) and WLAN (11000 Kbps/10 ms, 54000 Kbps/2 ms) is used in our experiments. In the first two experiments, the BDP of the access links held constant while the BDP of the access links in the second experiment is higher. In the second class of experiments the BDP of the two access links involved in a handoff differ. Here we perform two sets of experiments in which the BDP of the access links in the second set is higher than that in the first set. In each of the experiments we study the behaviour of TCP flows for the case of one, two and four long, simultaneous TCP flows.

4.2 Handoff from a Fast link to a Slow link

In this section we describe a set of experiments where a handoff occurs from a fast link to a slow link. As the problems that occur with a make-before-break handoff differ from that of a break-before-make handoff we discuss the two cases separately.

4.2.1 Make-Before-Break Handoff

The main problems of TCP in a make-before-break handoff from a fast link to a slow link are the occurrence of spurious RTOs and unnecessary retransmissions associated with them in addition to the packet drops due to a decrease in bandwidth. Our experiments described here show that the algorithms designed to avoid the spurious RTOs and to reduce the packet losses that are described in Section 3 are effective and improve the performance of TCP.

First we consider a handoff occurring between same BDP links even though the bandwidth and delay of the new access link differs considerably from that of the old link. The BDP of both access links is 10 packets. The problems in this case are the occurrence spurious RTOs and unnecessary retransmissions associated with them in addition to the packet drops due to a decrease in bandwidth.

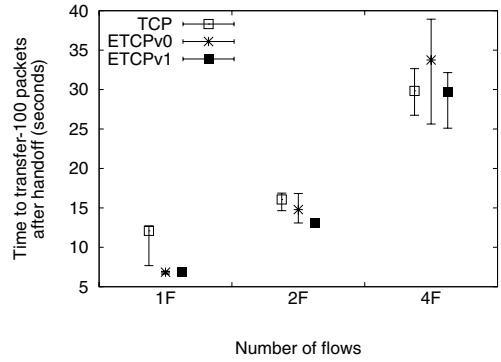


Figure 4: Time taken to transfer 100 packets after a make-before-break handoff from a 6400Kbps/9ms link to a 200Kbps/300 ms link

Figure 4 shows the time taken by regular TCP, ETCPv0 and ETCPv1 to transfer 100 packets after a make-before-break handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link. For a single TCP flow ETCPv1 and ETCPv0 show a 40 % reduction in transfer time compared to that of regular TCP. In the case of two flows a similar reduction in transfer time is seen whereas in the case of four flows the performance of regular TCP and ETCPv1 are nearly the same. The reason for this is that with the increase in the number of flows there is a consequent decrease in the size of the *cwnd* of each flow. As a result the typical problems of TCP in vertical handoff due to spurious RTOs and *cwnd* reduction do not have a significant impact on TCP performance. In the case of four flows, we can see that ETCPv0 performs worse than both regular TCP and ETCPv1 as ETCPv0 sets the *cwnd* to the BDP of the new link resulting in an aggressive behaviour leading to packet losses.

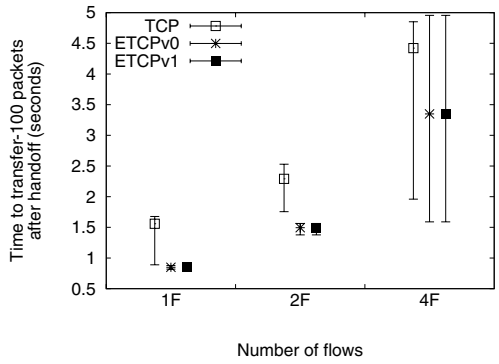


Figure 5: Time taken to transfer 100 packets after a make-before-break handoff from a 54000Kbps/2ms link to a 2000Kbps/50 ms link

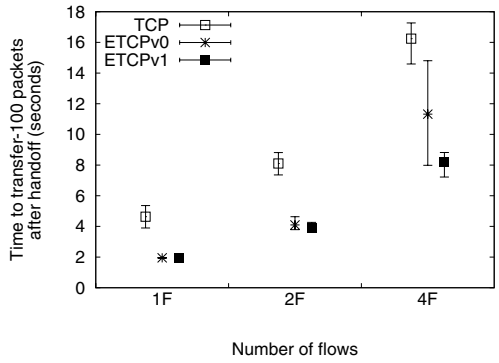


Figure 6: Time taken to transfer 100 packets after a make before-break handoff from a 11000Kbps/10ms link to a 700Kbps/75ms link

In this category of same BDP access link handoffs we next describe a handoff from a 54000 Kbps/2 ms link to a 2000 Kbps/50 ms link. Here the BDP of the links is set to a higher value (18 packets) and a higher bandwidth and lower delay compared to the corresponding values in experiment described above. Figure 5 shows that in the case of a single flow ETCPv1 and ETCPv0 reduce the transfer time for 100 packets after a handoff by about 40 % compared to regular TCP. We can also see from this figure that for two and four flows ETCPv1 still shows up to 30 % improvement in transfer time over regular TCP. With the increase in BDP, the *cwnd* for each flow increases resulting in packet losses that occur due to a make-before-break handoff. ETCPv1 improves the performance over regular TCP as it avoids packet losses due to spurious RTOs and *cwnd* reduction.

In the second class of experiments, the BDP of the access links differ. We have two sets of experiments in this class, namely, a handoff from a 11000 Kbps/10 ms link to a 700 Kbps/75 ms link (BDP of 18 and 9 packets respectively) and a handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link (BDP of 50 and 38 packets respectively).

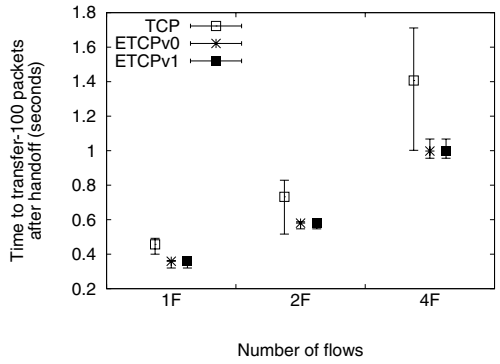


Figure 7: Time taken to transfer 100 packets after a make-before-break handoff from a 54000Kbps/4ms link to a 6000Kbps/50ms link

The main problem of regular TCP in a make-before-break handoff from a 11000 Kbps/10 ms link to a 700 Kbps/75 ms link is the large number of packet losses due to the reduction in BDP (BDP reduction from 18 packets to 9 packets) and also due to large decrease (about 15 times) in bandwidth. RTO recovery is needed to recover the lost packets and more losses may occur before regular TCP adapts itself to the *cwnd* of the new path. On the other hand, ETCPv1 sets the *cwnd* based on the BDP of the new link and the number of flows and is able to avoid the packet losses. Figure 6 illustrates the transfer time taken for regular TCP and ETCPs in this scenario. In the case of a single TCP flow, with ETCPv1 and ETCPv0, there is approximately 60 % reduction in transfer time of regular TCP.

When there are two TCP flows, packet losses lead to RTO recovery in the case of regular TCP but there are no packet losses for ETCPv1. The reduction in transfer time for ETCPv1 is around 50 % as the available bandwidth for a single flow is halved. When the number of flows increases to four, the available bandwidth share of the flows decreases. Regular TCP suffers packets losses and needs RTO recovery, while with ETCPv1 there are no packet losses as it sets the *cwnd* and *ssthresh* to the new link BDP scaled by the number of flows. ETCPv1 is able to reduce the transfer time by about 50 % compared to regular TCP. Figure 6 shows that ETCPv0 reduces the transfer time of regular TCP by about 35 % but there are still packet losses as it sets the *cwnd* and *ssthresh* to the new link BDP which is larger than the BDP corresponding to the flow's share of bandwidth when there are four simultaneous flows.

In a make-before-break handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link, both the access links have sufficiently high BDP values of 50 packets and 38 packets respectively. Figure 7 shows that ETCPv1 shows a 20-40 % reduction in the transfer time compared to the regular TCP even when there are four simultaneous TCP flows. This significant improvement in performance shows the effectiveness of our algorithms when the *cwnd* is sufficiently large.

4.2.2 Break-Before-Make Handoff

Next we describe the experiments involving a break-before-make handoff from a fast link to a slow link. Here we carry out the same set of experiments as in the case of make-before-break handoff described earlier.

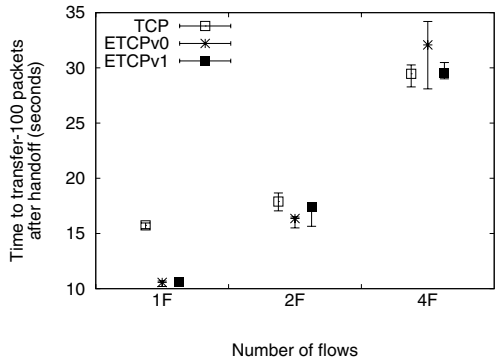


Figure 8: Time taken to transfer 100 packets after a break-before-make handoff from a 6400Kbps/9ms link to a 200Kbps/300ms link

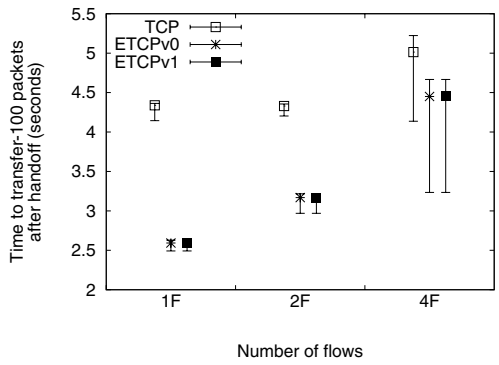


Figure 9: Time taken to transfer 100 packets after a break-before-make handoff from a 54000 Kbps/2ms link to a 2000 Kbps/50 ms link, disconnection period 1s

A break-before-make handoff results in packet losses and unused connection time. The algorithm in Figure 2 retransmits the first unacknowledged packet immediately if TCP is already in RTO recovery when a handoff notification arrives and this helps to utilize the connection as soon as the new access link is up after a handoff. The proper setting of the *ssthresh* by the algorithm helps in avoiding unnecessary reduction of *ssthresh* due to repeated timeouts.

Figure 8 shows the time taken for transferring 100 packets by the three TCP versions in a break-before-make handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link. We can see that in the case of a single flow both ETCPv0 and ETCPv1 reduce the transfer time by about 40 % compared to regular TCP. As the number of flows increases to four, the *cwnd* available for a single flow decreases and this reduces the number of packet losses due to a disconnection . In the case of four flows, we observe that ETCPv0 incurs additional losses resulting in increased transfer time compared to regular TCP and ETCPv1.

The marked improvement in the performance of the ETCPv1 over the regular TCP in a break-before-make handoff from a 54000 Kbps/2 ms link to a 2000 Kbps/50 ms link can be clearly seen from

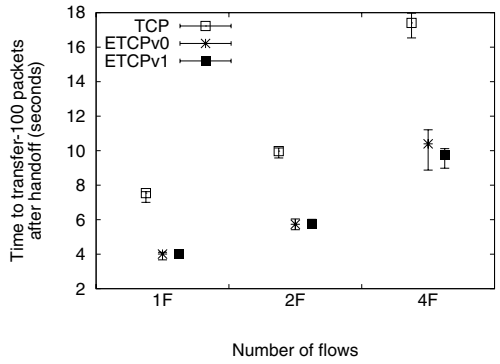


Figure 10: Time taken to transfer 100 packets after a break-before-make handoff from a 11000 Kbps/10ms link to a 700 Kbps/75 ms link, disconnection period 1s

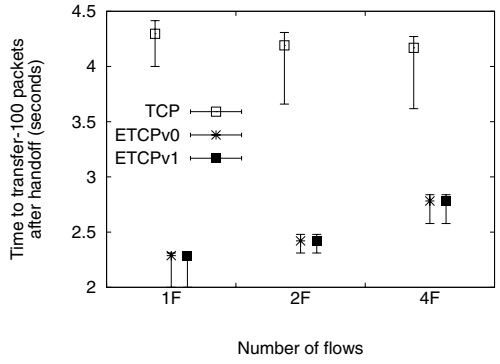


Figure 11: Time taken to transfer 100 packets after a break-before-make handoff from a 54000 Kbps/4ms link to a 6000 Kbps/50 ms link, disconnection period 1s

the Figure 9. Figure 10 shows that for all the flows, ETCPv1 reduces the transfer time by about 50 % compared to regular TCP. Figure 11 shows that ETCPv1 is effective in a break-before-make handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link. As the *cwnd* is sufficiently large ETCPv1 performs better than regular TCP when there are four simultaneous TCP flows. We can also see from this figure that there is about 20-40 % reduction in transfer time with ETCPv1 compared to regular TCP for all the flows.

4.3 Handoff from a Slow link to a Fast link

In this section we describe a set of experiments where both make-before-break and break-before-make handoffs occur from a slow link to a fast link.

4.3.1 Make-Before-Break Handoff

Packet reordering is the main problem of TCP in a make-before-break handoff from a slow link to a fast link. After a handoff, packets through the fast new link may arrive at the receiver sooner than the packets sent before the handoff through the slow old link resulting in packet reordering. Our algorithm given in Figure 3 is designed to mitigate the problems arising from packet reordering.

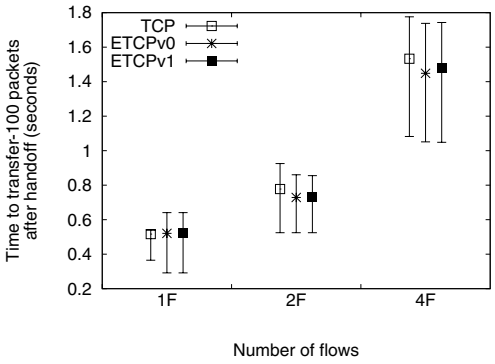


Figure 12: Time taken to transfer 100 packets after a make-before-break handoff from a 200Kbps/300ms link to a 6400Kbps/9ms link

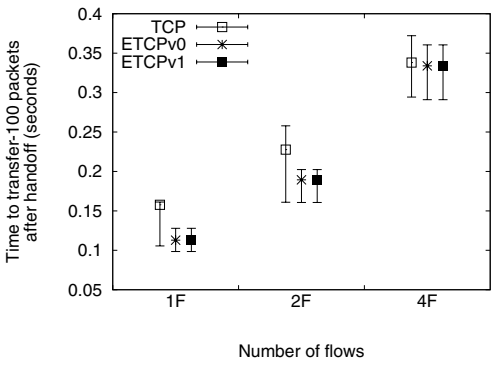


Figure 13: Time taken to transfer 100 packets after a make-before-break handoff from a 2000Kbps/50ms link to a 54000Kbps/2ms link

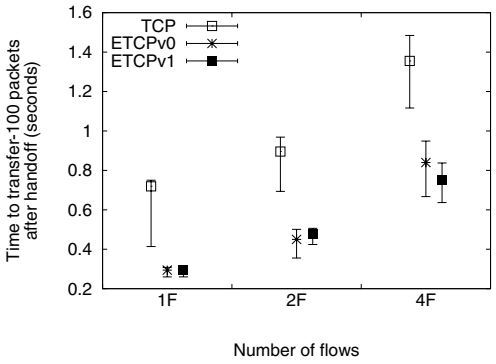


Figure 14: Time taken to transfer 100 packets after make-before-break handoff from a 700Kbps/75ms link to a 11000Kbps/10ms link

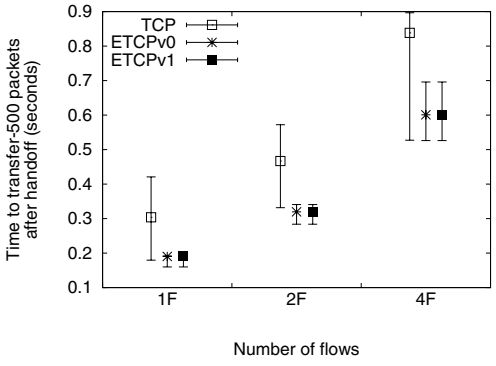


Figure 15: Time taken to transfer 500 packets after make-before-break handoff from a 6000Kbps/50ms link to a 54000Kbps/4ms link

Figure 12 shows the results of a make-before-break handoff from a 200 Kbps/300 ms link to a 6400 Kbps/9 ms link. Even though the transfer time of ETCPv1 and TCP are quite the same, we have observed in our experiments that ETCPv1 reduces the unnecessary retransmissions caused by packet reordering.

Figure 13 shows the results for a make-before-break handoff from a 2000 Kbps/50 ms link to a 54000 Kbps/2 ms link. In the cases of one and two TCP flows our algorithm in Figure 3 reduces the transfer time taken by regular TCP by about 30 %. In the case of four flows, the *cwnd* for a single flow is relatively small and the unnecessary retransmissions and *cwnd* reduction due packet reordering have only a minor effect on TCP performance. We can see from Figure 13 that when there are four simultaneous TCP flows sharing the link at the time of handoff, ETCPv1 and regular TCP have comparable performance. In our experiments we observed that ETCPv1 is effective in reducing the unnecessary retransmissions caused by packet reordering.

Figure 14 shows the results of the transfer time taken by regular TCP and ETCPv1 in a handoff from a 700 Kbps/75 ms link to a 11000 Kbps/10 ms link. Here the handoff is from a low BDP link to a high BDP link. ETCPv1 shows an improved performance, more than 50 % reduction in transfer time of regular TCP for all the flows.

Fig 15 shows the time taken to transfer 500 packets after a make-before-break handoff from a 6000 Kbps/50 ms link to a 54000 Kbps/9 ms link. Packet reordering is again the main problem of TCP in this scenario. ETCPv1 utilizes the new high bandwidth link effectively and is able to transfer 300-400 packets while waiting for the packets sent earlier through the old link. Here we have taken the time to transfer 500 packets after a handoff which is an adequate time in this scenario for TCP to recover from the effects of a handoff. ETCPv1 is able to reduce the transfer time of regular TCP by about 30 % in this scenario.

4.3.2 Break-Before-Make Handoff

In an break-before-make handoff, our algorithm in Figure 2 immediately retransmits the lost segment and thereby helps to utilize the fast link after the handoff.

As shown in Figure 16 for a break-before-make handoff from a 200 Kbps/300 ms link to a 6400 Kbps/9 ms link the transfer time

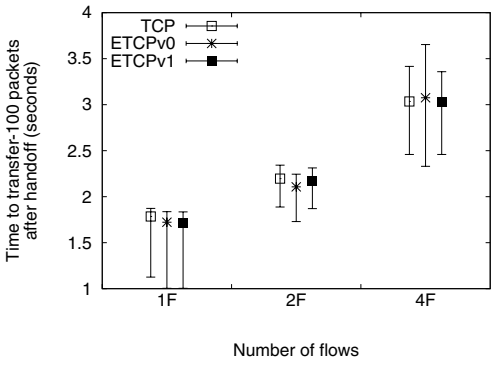


Figure 16: Time taken to transfer 100 packets after a break-before-make handoff from a 200Kbps/300ms link to a 6400Kbps/9mslink, disconnection period 1s

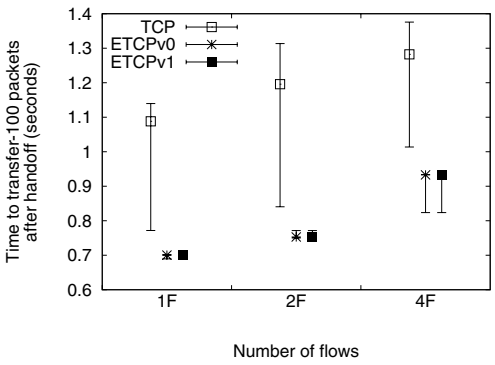


Figure 17: Time taken to transfer 100 packets after a break-before-make handoff from a 2000Kbps/50ms link to a 54000 Kbps/2ms link, disconnection period 1s

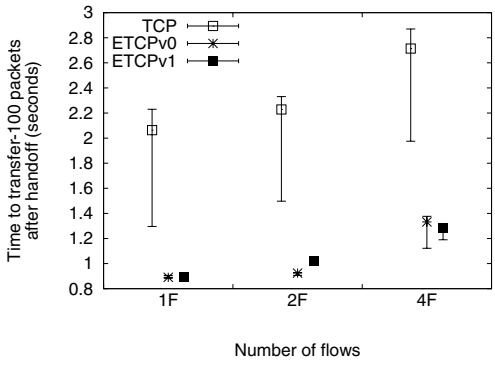


Figure 18: Time taken to transfer 100 packets after a break-before-make handoff from a 700Kbps/75ms link to a 11000Kbps/10ms link, disconnection period 1s

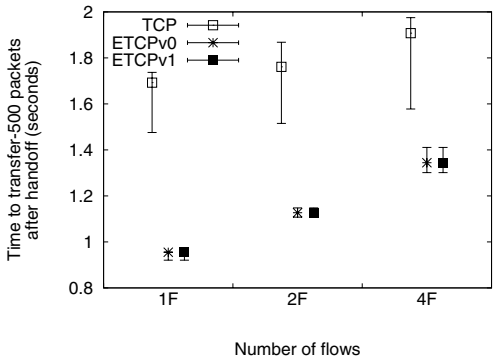


Figure 19: Time taken to transfer 500 packets after a break-before-make handoff from a 6000Kbps/50ms link to a 54000Kbps/4ms link, disconnection period 1s

of ETCPv1 and regular TCP are equal for all the flows. This is because the RTO value of the old access link (2.5 seconds to 3 seconds) is larger than the disconnection period of one second and a timeout will not occur during the disconnection time. Therefore ETCPv1 and ETCPv0 will not enter the algorithm given in Figure 2. When a disconnection period is four seconds or longer the algorithm in Figure 2 will be entered and consequently there will be significant improvement arising from its use.

We can observe from Figure 17 that ETCPv1 shows nearly 50 % reduction in transfer time over regular TCP when a break-before-make handoff occurs from a 2000 Kbps/50 ms link to a 54000 Kbps/2 ms link. The reason for this improvement is due to ability of ETCPv1 to utilize the high bandwidth link immediately after a handoff by using the algorithm in Figure 2 whereas regular TCP waits for the next RTO to start the transmission after the handoff. We can see from Figure 18 that in a handoff from a 700 Kbps/75 ms link to a 11000 Kbps/10 ms link ETCPv1 shows 50 % reduction in transfer time over regular TCP for all the flows.

Figure 19 shows the time taken for transferring 500 packets after a break-before-make handoff from a 6000 Kbps/50 ms link to a 54000 Kbps/4 ms link. We can see that ETCPv1 obtains over 50 % reduction in transfer time over regular TCP for all the flows we consider in our experiments. The reason for the improvement is the same as given in the previous paragraph.

5. CONCLUSIONS

In this paper we study the behaviour of multiple TCP flows in the presence of a vertical handoff. Through extensive simulations we show that the proposed cross-layer assisted algorithms, which utilize the information about the number of simultaneous TCP flows and the bandwidth and delay of the access links, are effective in avoiding the problems of TCP due to a vertical handoff and improve TCP performance. The problems of TCP in a vertical handoff due to the number of unnecessary retransmissions and packet losses are aggravated with the increase in the size of the *cwnd*. With the increase in the number of TCP flows the size of the *cwnd* decreases roughly in inverse proportion to the number of TCP flows that share the bottleneck access link. Consequently, as the number of simultaneous TCP flows increases, the typical problems of TCP due to spurious RTOs, packet reordering and *cwnd* reduction

arising from a vertical handoff tend to have diminishing impact on TCP performance, in particular if the *cwnd* is small. However, if the *cwnd* is sufficiently large the algorithms proposed in this paper will be effective for multiple TCP flows in various vertical handoff scenarios.

Acknowledgments

This work has been carried out in the frame of the WISEciti research project. We would like to thank our colleagues in the Wireless Internet group and the Wiseciti consortium for discussions and support in carrying out this research.

6. REFERENCES

[1] 3GPP. High Speed Downlink Packet Access (HSDPA); Overall UTRAN description. Technical Report 3GPP TS 25.855, Mar. 2002.

[2] M. Allman and A. Falk. On the Effective Evaluation of TCP. *ACM Computer Communication Review*, 5(29):59 – 70, Oct. 1999.

[3] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM Computer Communication Review*, 32(1):20 – 30, Jan. 2002.

[4] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP. IETF RFC 3517, Apr. 2003.

[5] D. Borman, R. Braden, and V. Jacobson. TCP Extensions for High Performance. IETF RFC 1323, May 1992.

[6] L. Daniel. TCP Performance with Vertical Handoff. Licentiate Thesis, Series of Publications C, No. C-2008-221, Nov. 2008. Available at: http://www.cs.helsinki.fi/u/ldaniel/lic-thesis-tcp-vho.pdf.

[7] L. Daniel, I. Järvinen, and M. Kojo. Combating Packet Reordering in Vertical Handoff Using Cross-Layer Notifications to TCP. In *Proc. IEEE Conference on Wireless and Mobile Computing, (WiMob08)*, Oct. 2008.

[8] L. Daniel and M. Kojo. Adapting TCP for Vertical Handoffs in Wireless Networks. In *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, pages 151 – 158, Nov. 2006.

[9] L. Daniel and M. Kojo. Employing cross-layer assisted tcp algorithms to improve tcp performance with vertical handoffs. *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, 1(4/5/6):433–465, 2008.

[10] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 3782, Apr. 2004.

[11] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgment (SACK) Option for TCP. IETF RFC 2883, July 2000.

[12] Y. Gou, D. Pearce, and P. Mitchell. A Receiver-based Vertical Handover Mechanism for TCP Congestion Control. *IEEE Transactions on Wireless Communications*, 5(10):2824–2833, Oct. 2006.

[13] A. Gurtov and J. Korhonen. Effect of Vertical Handovers on Performance of TCP-Friendly Rate Control. *ACM Mobile Computing and Communications Review*, 8(3):73–87, July 2004.

[14] W. Hansmann and M. Frank. On Things to Happen During a TCP Handover. In *Proc. 28th IEEE Conference on Local Computer Networks (LCN'03)*, pages 109– 118, Oct. 2003.

[15] H. Huang and J. Cai. Improving TCP Performance during Soft Vertical Handoff. In *Proc. 19th International Conference on Advanced Information Networking and Applications (AINA'05)*, volume 2, pages 329–332, Mar. 2005.

[16] IEEE. IEEE standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. ANSI/IEEE Std 802.11. 1999 Edition (R 2003), 2003.

[17] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of ACM SIGCOMM '91*, pages 3–15, 1991.

[18] S.-E. Kim and J. A. Copeland. TCP for Seamless Vertical Handoff in Hybrid Mobile Data Networks. In *Proc. IEEE Globecom 2003*. IEEE, Dec. 2003.

[19] D. Li, K. Sleurs, E. V. Lil, and A. V. de Capelle. A fast adaptation mechanism for TCP vertical handover. In *Proc. of the International Conference on Advanced Technologies for Communications, ATC 2008*, Oct. 2008.

[20] Y. Lin and H. Chang. VA-TCP: A Vertical Handoff-Aware TCP. In *Proceedings of the ACM symposium on Applied computing*, pages 237–238. ACM, 2007.

[21] R. Ludwig and R. H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communication Review*, 30(1):30 – 36, Jan. 2000.

[22] J. Manner and M. Kojo. Mobility Related Terminology. IETF RFC 3753, June 2004.

[23] D. Molkdar, W. Featherstone, and S. Lambotharan. An Overview of EGPRS: the packet data component of EDGE. *Electronics and Communication engineering Journal*, 14:21–38, Feb. 2002.

[24] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. IETF RFC 2988, Nov. 2000.

[25] J. Postel. Transmission Control Protocol. IETF RFC 793, Sept. 1981.

[26] H. Rutagemwa, S. Pack, X. Shen, and J. W. Mark. Robust cross-layer design of wireless-profiled tcp mobile receiver for vertical handover. *IEEE Transactions on Vehicular Technology*, 56(6):3899–3911, Nov. 2007.

[27] P. Sarolahti, M. Allman, and S. Floyd. Determining an Appropriate Sending Rate Over an Underutilized Network Path. *Computer Networks (Elsevier)*, pages 1815–1832, May 2007.

[28] P. Sarolahti, M. Kojo, and K. Raatikainen. F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts. *ACM SIGCOMM Computer Communication Review*, 33(2):51–63, Apr. 2003.

[29] P. Sarolahti, J. Korhonen, L. Daniel, and M. Kojo. Using Quick-Start to Improve TCP Performance with Vertical Hand-offs. In *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, pages 897–904, Nov. 2006.

[30] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol Enhancements for Intermittently Connected Hosts. *ACM Computer Communication Review*, 35(2):5–18, July 2005.

[31] M. Stemm and R. H. Katz. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

[32] K. Tsukamoto, Y. Fukuda, Y. Hori, and Y. Oie. New TCP Congestion Control Scheme for Multimodal Mobile Hosts. *IEICE Transactions on Communications*, E89-B(6):1825–1836, 2006.

[33] WiMAX. Worldwide Interoperability for Microwave Access (WiMAX). www.wimaxforum.org.

Errata for Paper 4

1. Page 21 last paragraph of the left column, 38 and 50 packets instead of 50 and 38 packets
2. Page 21 last but one paragraph right column, 38 and 50 packets instead of 50 and 38 packets
3. Page 23, the 3rd paragraph from bottom, the sentence Fig 15 shows ... 54000 Kbps/4 ms instead of 54000 Kbps/9 ms
4. Page 24, in the figures Fig 16, 17, 18 and 19 the disconnection period is 500 ms instead of 1 second

TIETOJENKÄSITTELYTIETEEN LAITOS
PL 68 (Gustaf Hällströmin katu 2 b)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Gustaf Hällströmin katu 2 b)
FIN-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**

Reports may be ordered from: Kumpula Science Library, P.O. Box 64, FIN-00014 University of Helsinki, FINLAND.

A-2004-1 M. Koivisto: Sum-product algorithms for the analysis of genetic risks. 155 pp. (Ph.D. Thesis)

A-2004-2 A. Gurtov: Efficient data transport in wireless overlay networks. 141 pp. (Ph.D. Thesis)

A-2004-3 K. Vasko: Computational methods and models for paleoecology. 176 pp. (Ph.D. Thesis)

A-2004-4 P. Sevon: Algorithms for Association-Based Gene Mapping. 101 pp. (Ph.D. Thesis)

A-2004-5 J. Viljamaa: Applying Formal Concept Analysis to Extract Framework Reuse Interface Specifications from Source Code. 206 pp. (Ph.D. Thesis)

A-2004-6 J. Ravantti: Computational Methods for Reconstructing Macromolecular Complexes from Cryo-Electron Microscopy Images. 100 pp. (Ph.D. Thesis)

A-2004-7 M. Kääriäinen: Learning Small Trees and Graphs that Generalize. 45+49 pp. (Ph.D. Thesis)

A-2004-8 T. Kivioja: Computational Tools for a Novel Transcriptional Profiling Method. 98 pp. (Ph.D. Thesis)

A-2004-9 H. Tamm: On Minimality and Size Reduction of One-Tape and Multitape Finite Automata. 80 pp. (Ph.D. Thesis)

A-2005-1 T. Mielikäinen: Summarization Techniques for Pattern Collections in Data Mining. 201 pp. (Ph.D. Thesis)

A-2005-2 A. Doucet: Advanced Document Description, a Sequential Approach. 161 pp. (Ph.D. Thesis)

A-2006-1 A. Viljamaa: Specifying Reuse Interfaces for Task-Oriented Framework Specialization. 285 pp. (Ph.D. Thesis)

A-2006-2 S. Tarkoma: Efficient Content-based Routing, Mobility-aware Topologies, and Temporal Subspace Matching. 198 pp. (Ph.D. Thesis)

A-2006-3 M. Lehtonen: Indexing Heterogeneous XML for Full-Text Search. 185+3 pp. (Ph.D. Thesis)

A-2006-4 A. Rantanen: Algorithms for ^{13}C Metabolic Flux Analysis. 92+73 pp. (Ph.D. Thesis)

A-2006-5 E. Terzi: Problems and Algorithms for Sequence Segmentations. 141 pp. (Ph.D. Thesis)

A-2007-1 P. Sarolahti: TCP Performance in Heterogeneous Wireless Networks. (Ph.D. Thesis)

A-2007-2 M. Raento: Exploring privacy for ubiquitous computing: Tools, methods and experiments. (Ph.D. Thesis)

A-2007-3 L. Aunimo: Methods for Answer Extraction in Textual Question Answering. 127+18 pp. (Ph.D. Thesis)

A-2007-4 T. Roos: Statistical and Information-Theoretic Methods for Data Analysis. 82+75 pp. (Ph.D. Thesis)

A-2007-5 S. Leggio: A Decentralized Session Management Framework for Heterogeneous Ad-Hoc and Fixed Networks. 230 pp. (Ph.D. Thesis)

A-2007-6 O. Riva: Middleware for Mobile Sensing Applications in Urban Environments. 195 pp. (Ph.D. Thesis)

A-2007-7 K. Palin: Computational Methods for Locating and Analyzing Conserved Gene Regulatory DNA Elements. 130 pp. (Ph.D. Thesis)

A-2008-1 I. Autio: Modeling Efficient Classification as a Process of Confidence Assessment and Delegation. 212 pp. (Ph.D. Thesis)

A-2008-2 J. Kangasharju: XML Messaging for Mobile Devices. 24+255 pp. (Ph.D. Thesis).

A-2008-3 N. Haiminen: Mining Sequential Data – in Search of Segmental Structures. 60+78 pp. (Ph.D. Thesis)

A-2008-4 J. Korhonen: IP Mobility in Wireless Operator Networks. (Ph.D. Thesis)

A-2008-5 J.T. Lindgren: Learning nonlinear visual processing from natural images. 100+64 pp. (Ph.D. Thesis)

A-2009-1 K. Hätönen: Data mining for telecommunications network log analysis. 153 pp. (Ph.D. Thesis)

A-2009-2 T. Silander: The Most Probable Bayesian Network and Beyond. (Ph.D. Thesis)

A-2009-3 K. Laasonen: Mining Cell Transition Data. 148 pp. (Ph.D. Thesis)

A-2009-4 P. Miettinen: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. 164+6 pp. (Ph.D. Thesis)

A-2009-5 J. Suomela: Optimisation Problems in Wireless Sensor Networks: Local Algorithms and Local Graphs. 106+96 pp. (Ph.D. Thesis)

A-2009-6 U. Köster: A Probabilistic Approach to the Primary Visual Cortex. 168 pp. (Ph.D. Thesis)

A-2009-7 P. Nurmi: Identifying Meaningful Places. 83 pp. (Ph.D. Thesis)

A-2009-8 J. Makkonen: Semantic Classes in Topic Detection and Tracking. 155 pp. (Ph.D. Thesis)

A-2009-9 P. Rastas: Computational Techniques for Haplotype Inference and for Local Alignment Significance. 64+50 pp. (Ph.D. Thesis)

A-2009-10 T. Mononen: Computing the Stochastic Complexity of Simple Probabilistic Graphical Models. 60+46 pp. (Ph.D. Thesis)

A-2009-11 P. Kontkanen: Computationally Effcient Methods for MDL-Optimal Density Estimation and Data Clustering. 75+64 pp. (Ph.D. Thesis)

A-2010-1 M. Lukk: Construction of a global map of human gene expression - the process, tools and analysis. 120 pp. (Ph.D. Thesis)

A-2010-2 W. Hämäläinen: Efficient search for statistically significant dependency rules in binary data. 163 pp. (Ph.D. Thesis)

A-2010-3 J. Kollin: Computational Methods for Detecting Large-Scale Chromosome Rearrangements in SNP Data. 197 pp. (Ph.D. Thesis)

A-2010-4 E. Pitkänen: Computational Methods for Reconstruction and Analysis of Genome-Scale Metabolic Networks. 115+88 pp. (Ph.D. Thesis)

A-2010-5 A. Lukyanenko: Multi-User Resource-Sharing Problem for the Internet. 168 pp. (Ph.D. Thesis)