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.

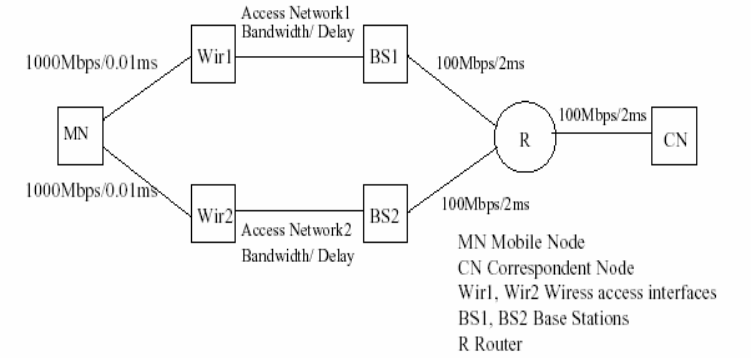
As there may be a significant change in link characteristics due to a vertical handoff, Sarolahti et al. (2006) study the suitability of Quick-Start (Sarolahti et al., 2007a) to set the *cwnd* and *ssthresh* after a vertical handoff. An explicit cross-layer handoff notification is employed to trigger 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.*

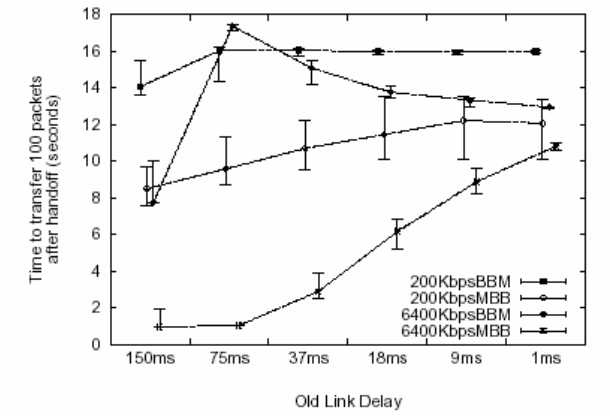
Given that the delay increase is significant, the small RTO value based on the measurements over the low-delay path causes the TCP retransmission timer to expire spuriously as the ACKs take the high-delay link after the handoff resulting in a significant increase in RTT. Typically no packet losses occur 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

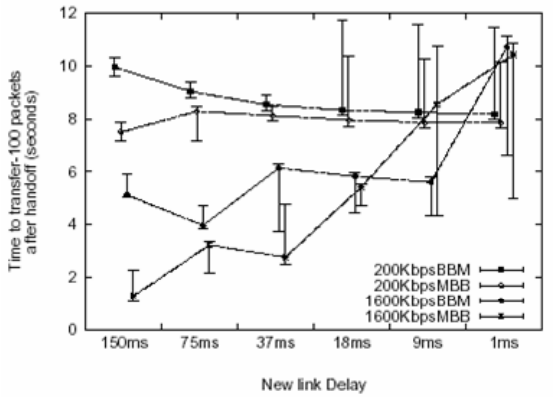
2 the ACKs arriving through the new link triggering more packets to be sent to the low BDP link resulting in packet losses

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.

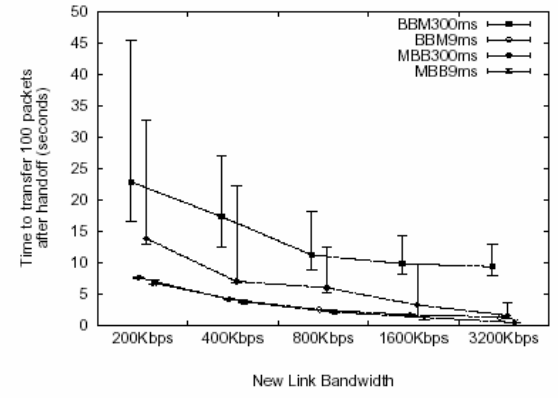
4.2 TCP behaviour due to changes in bandwidth

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)



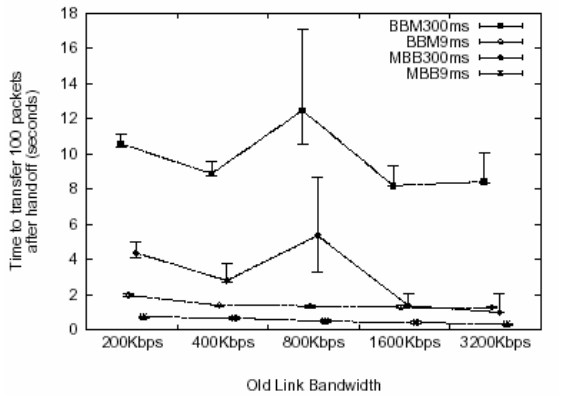
Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff between same delay links with varying new link bandwidth and the bandwidth of the old link fixed at 6400 Kbps.

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.

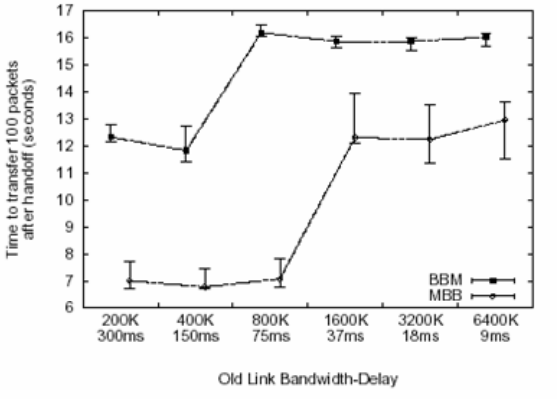
TCP behaviour for make-before-break handoff between 9 ms delay links is similar to the behaviour we described for 300 ms links. In the case of break-before-make handoff, TCP will be in RTO recovery during the disconnection period and another RTO is required to recover the losses. As explained in Section 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.

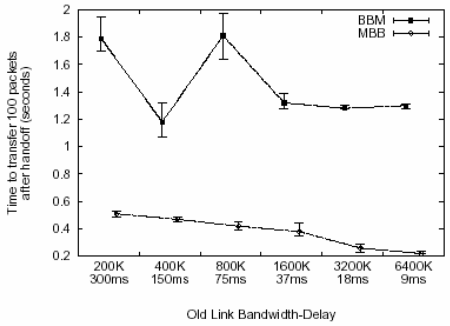
Figure 6 Handoff from a high-bandwidth/low-delay link to a low-bandwidth/high-delay link with fixed BDP



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

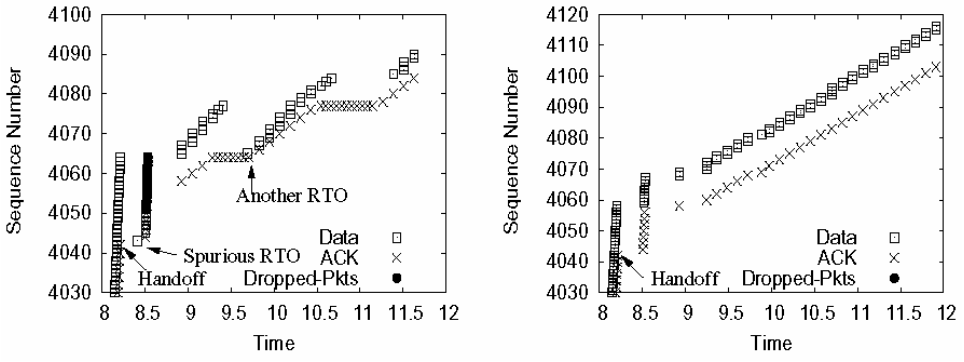
2 TCP is in fast recovery when the handoff occurs

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



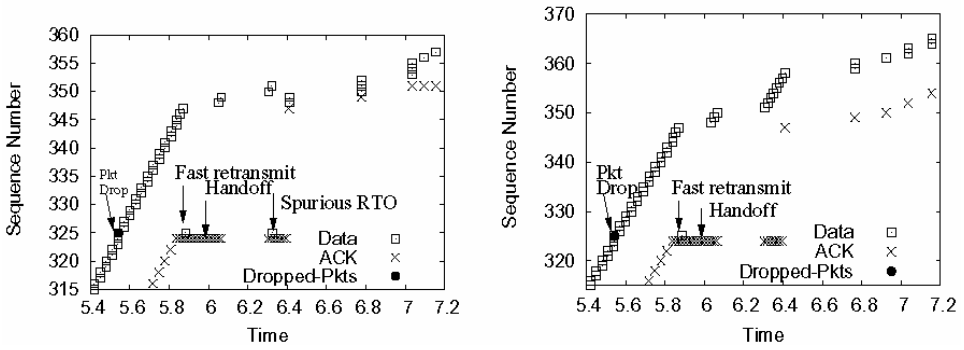
In Sections 4.2 and 4.3, we observed that spurious RTOs occur in more than 85–90% of the handoff points when the delay of the new link is at least eight times that of the old link. A typical scenario giving the time-sequence graph of a make-before-break handoff from a 6400 Kbps/9ms link to a 200 Kbps/300 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

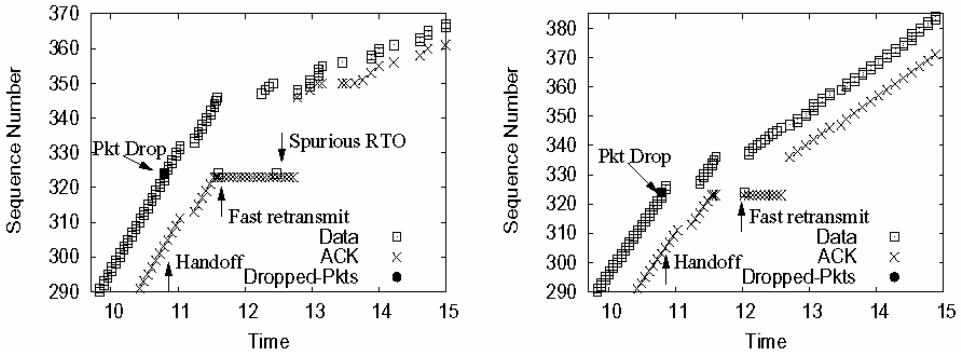


Case 3

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.

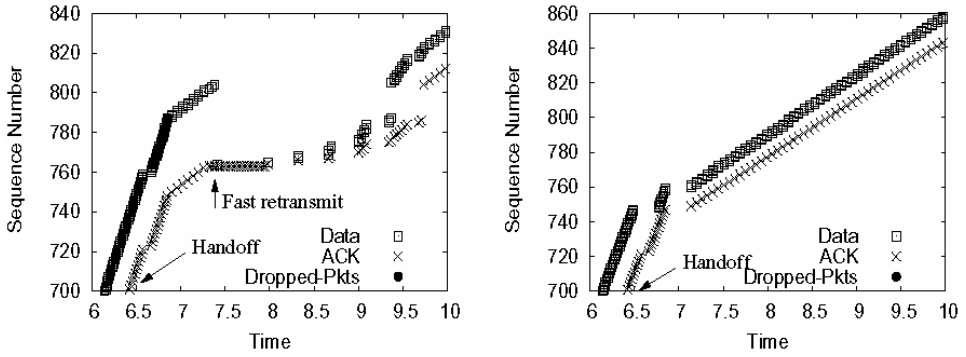
For Case 3, Figure 10a shows that there are 22 segments outstanding when TCP fast retransmits. The arrival of the ACK for the fast retransmitted packet takes 1.17 seconds which is more than the RTO value of the regular TCP (850 ms) at that point and spurious RTO occurs due to the late arrival of the 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 ($FlightSize > 2 * BDP_{newlink}$)
    $cwnd = max(2, BDP_{newlink})$
    $ssthresh = cwnd$
    $cwnd\_reduced = 1$

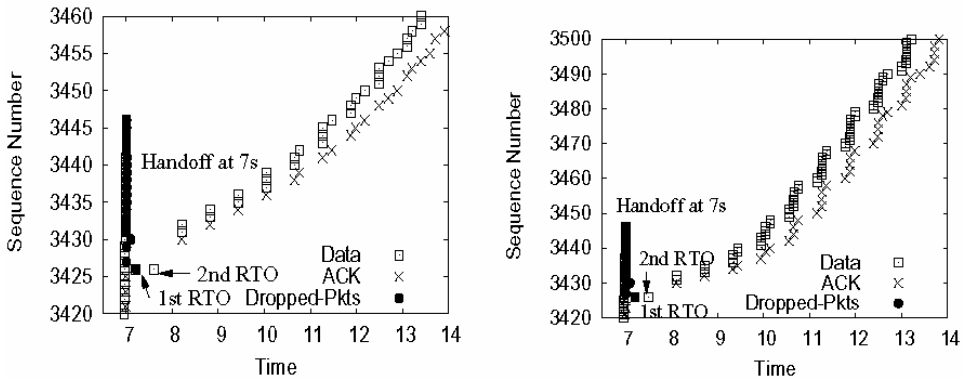
A comparison of Figure 12a and Figure 12b shows the effect of the algorithm A2. The *FlightSize* when the handoff notification arrives is 39 packets whereas the BDP of the new link is only 11 packets. As this *FlightSize* is larger than twice the BDP of the new link, the algorithm A2 sets the *ssthresh* 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
```

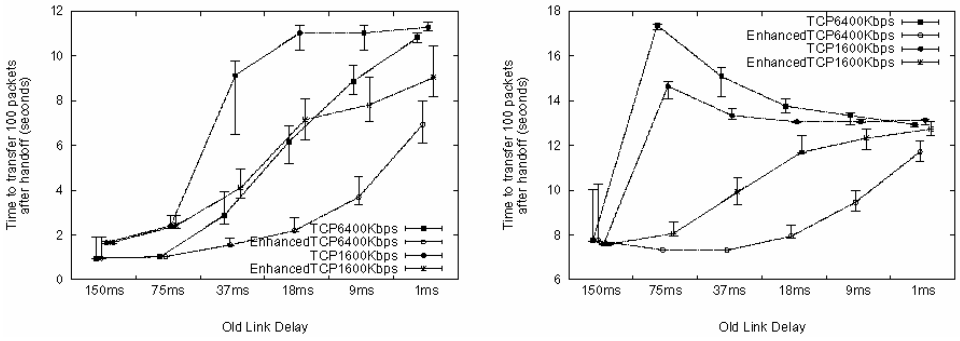
We designate the TCP sender algorithm incorporating the enhancements given by algorithms A1, A2, A3 and A4 as the enhanced TCP. In Section 6 we study the behaviour of the enhanced TCP in various handoff scenarios.

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



(a) (b)

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.

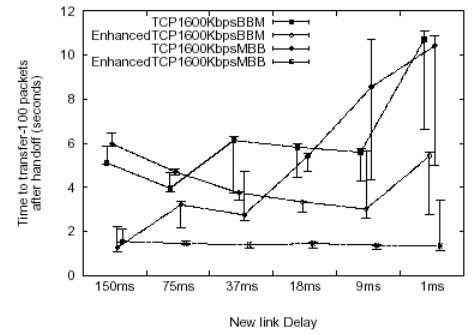
Figure 18 shows a comparison of the transfer time for regular TCP and enhanced TCP after make-before-break and break-before-make handoffs from a high-delay link to a low-delay link when both links have the same bandwidth of 1600 Kbps.

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

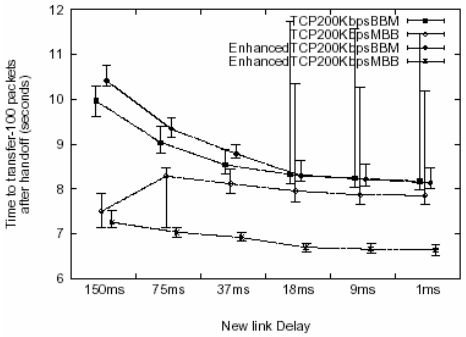
2 the slow convergence of RTO to the low delay of the new link.

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)

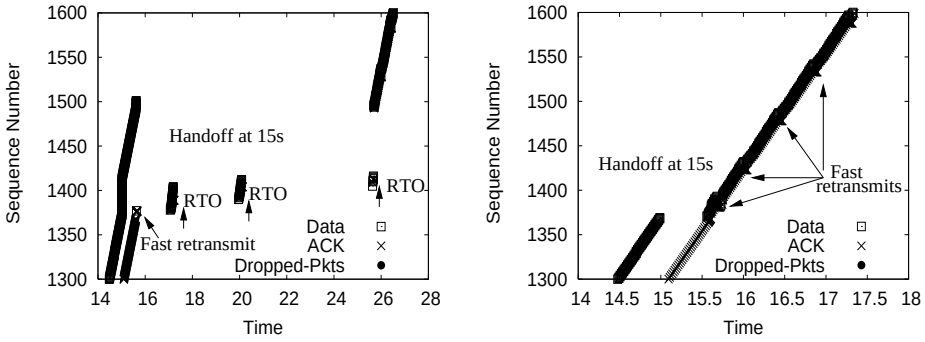


Note: Transfer time for 100 packets after a make-before-break (MBB) and a break-before-make (BBM) handoff between 200 Kbps links with varying new link delays and the delay of the old link is fixed at 300 ms.

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*

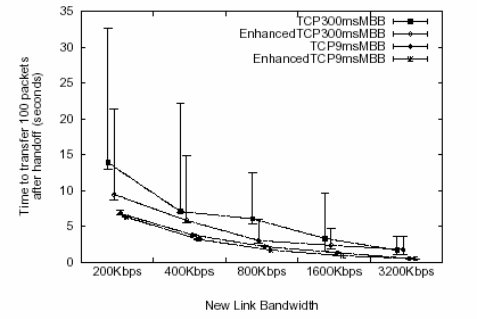
to BDP of the new link underutilises the path slightly resulting in about a 10% increase in the transfer time for enhanced TCP.

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.

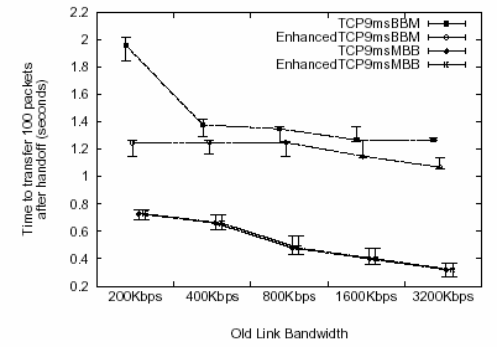
For the 300 ms delay link with regular TCP, a make-before-break handoff from a high-bandwidth to a low-bandwidth link results in congestion-related packet losses. The enhanced TCP by applying algorithm A2 sets *ssthresh* and *cwnd* to the BDP of the new link to avoid packet losses due to congestion and 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

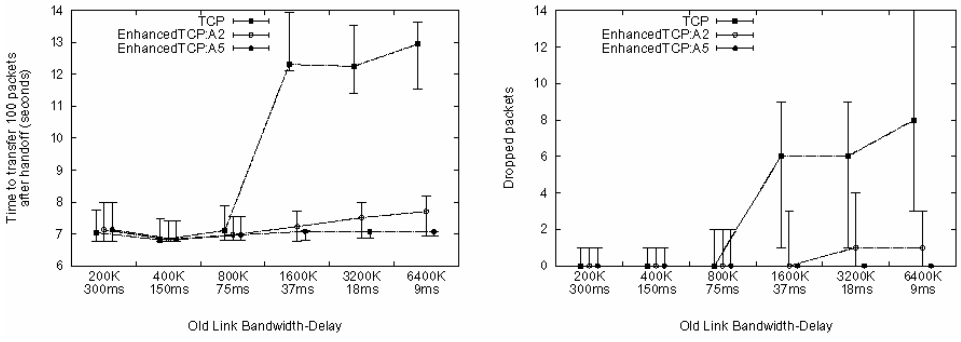
In this set of experiments we have two sets of fixed bandwidth/delay values namely, 200 Kbps/300 ms and 6400 Kbps/9 ms.

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



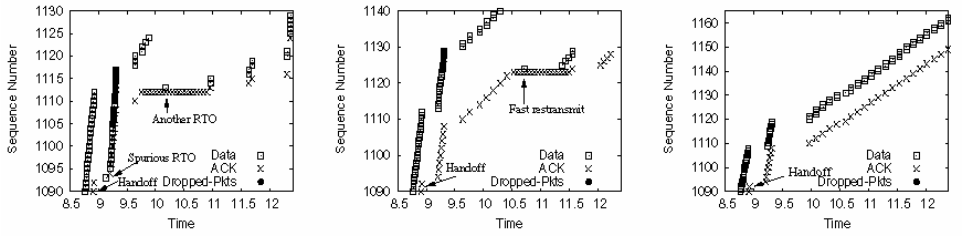
(a) (b)

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.

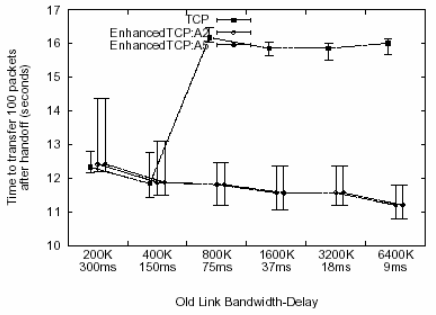
Figure 25 A5: Modified algorithm to reduce congestion related packet losses

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)



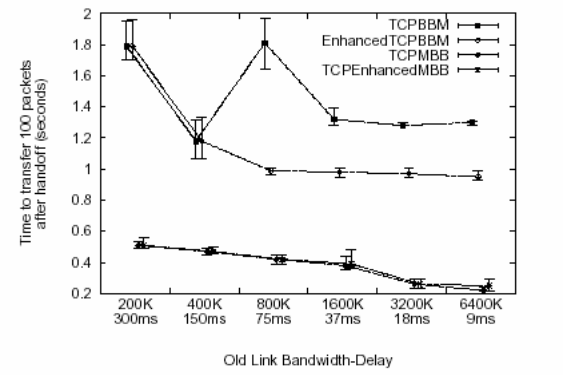
Note: Transfer time for 100 packets after a break-before-make handoff between the same BDP links with varying bandwidth and delay of the old link and the bandwidth and delay of the new link are fixed at 200Kbps/300ms. The disconnection period is 500 ms.

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.