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

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

Abstract

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

1. Introduction

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

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

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

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

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

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

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

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

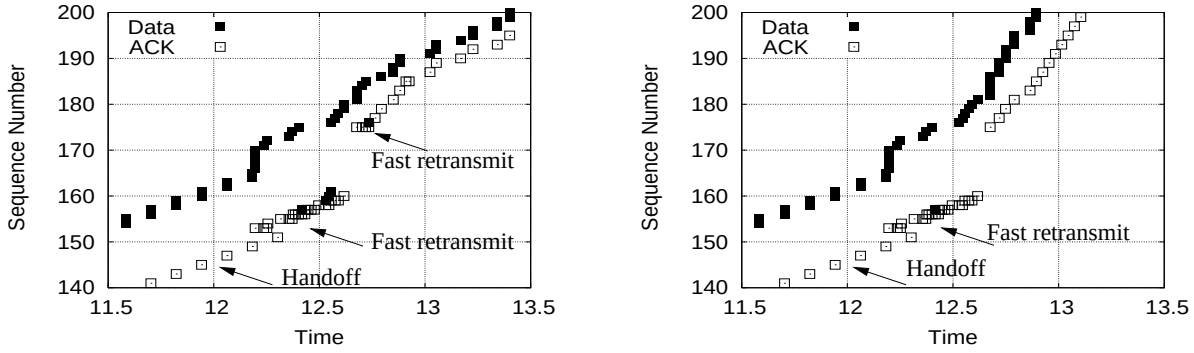
This paper reports a follow-up work on the research presented in our earlier papers [7, 8]. In those papers we described the various problems arising from a vertical handoff including packet reordering and proposed algorithms to mitigate the effect of some of these problems such as spurious RTOs, congestion-related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

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

2. Packet reordering due to vertical handoff

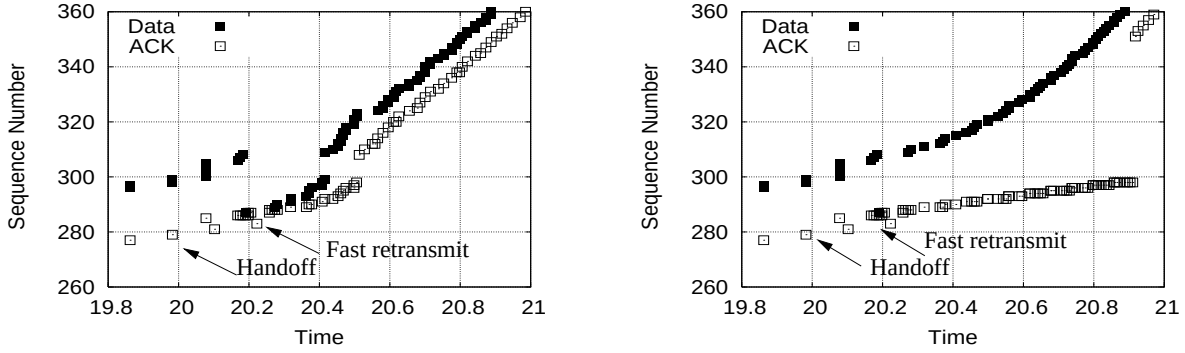
Packet reordering may occur when a make-before-break handoff occurs from a slow access link to a fast access link [10, 7]. During a make-before-break handoff, an MN can receive packets through the old link as well as through the new link. The sequence numbers of the packets arriving through the new link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

We study the effect of reordering due to a vertical handoff using ns-2 [17] simulations. The baseline TCP for our



(a) Regular TCP (b) Enhanced TCP

Figure 1: Comparison of regular TCP and enhanced TCP: make-before-break handoff at 12 sec from a 200 Kbps/300 ms link to a 800 Kbps/75 ms link.



(a) Regular TCP (b) Enhanced TCP

Figure 2: Comparison of regular TCP and enhanced TCP: make-before-break handoff at 20 sec from a 200 Kbps/300 ms link to a 1600 Kbps/37 ms link.

study is the TCP-Sack1 algorithm in ns-2 and we refer to it as regular TCP. We now describe the behaviour of regular TCP for two packet reordering scenarios where the ratio of the delays (or bandwidths) of the old and the new links are such that the arrival pattern of *dupacks* differ. The BDP of the links involved in a handoff is kept constant in order to isolate the effect of reordering.

Consider a handoff from a 200 Kbps/300 ms link to a 800 Kbps/75 ms link, the ratio of the link delays is 4 and the two links have the same BDP of 10 packets. An interesting behaviour of regular TCP is shown in Figure 1a. A handoff occurs at 12.00 sec, and 8 out-of-order packets are received but they are not consecutive as the packets sent before handoff through the old link arrive at the receiver in-between. As the first two groups of *dupacks* consist of only two of them, the *fast retransmit* is triggered only by the third *dupacks* group at 12.40 sec. The last packet through the old link arrives earlier than the retransmissions through the new link and the fast recovery completes at 12.65 sec. The unnecessary retransmissions through the new link generate dupacks and TCP invokes *fast retransmit* unnecessarily again at 12.75 sec. Thus TCP reduces the *cwnd* twice which brings down the sending rate. While the reordering and TCP entering *fast retransmit* are observed to occur nearly at all handoff points, the behaviour described above where TCP subsequently goes into *fast retransmit* is observed in 20 % of the handoff points.

Figure 2a shows a time-sequence graph for a handoff from a 200 Kbps/300 ms link to a 1600 Kbps/37 ms link. In scenario of the Figure 2a the handoff occurs at 20.00 sec and the last packet sent before handoff arrives at the receiver at about 900 ms after the handoff. All forward progress of the flow happens over the new link starting from a reordered ACK at 20.07 sec. Then six packets are received out of order and their *dupacks* trigger fast retransmit at 20.19 sec. 12 packets are unnecessarily retransmitted.

We observe that as the ratio of the link delays increases, the *fast retransmit* algorithm is triggered immediately after a handoff and the number of unnecessary retransmissions increases. Our algorithm described in Section 3 minimizes these unnecessary retransmissions.

3. The proposed algorithm

As the end-to-end path characteristics of a TCP connection over a fixed Internet can be assumed to be relatively stable over the lifetime of the connection, we regard the changes in the path characteristics as arising from the changes in the access link characteristics due to a vertical handoff. We calculate a set of parameters from the access link characteristics. These parameters are taken as a rough estimate of the link characteristics and the algorithm makes a conservative use of these values in order to ensure that the lack of accuracy in determining these parameters does not make the algorithm aggressive.

Cross-layer notifications have been shown to be beneficial to TCP when path characteristics change widely due to a vertical handoff [20, 19, 7, 8, 21]. Many evaluations have been made on how to deliver the link characteristics information [20, 12, 13, 18, 21]. In essence, the TCP layer at the MN can be locally notified of the changes in the local link characteristics during a vertical handoff. This information can be sent to the TCP layer at the CN as a TCP option or along with the mobility registration message such as the binding update message in Mobile IPv6 [11] to be further forwarded to the TCP layer. A TCP sender can interpret the notification from the lower layers as a hint about the characteristics of the end-to-end path and adjust the congestion control parameters and RTO estimate so as to adapt to the new path in an efficient and timely manner. But even though a very conservative TCP approach is selected like with the *Response Connectivity Change Indication* (RLCI) mechanism [21] which forces TCP to slow-start if the new network path is unknown, unnecessary retransmissions may occur in the case of make-before-break handoffs.

In this work we model that the MN can deliver the handoff notification piggybacked in the mobility signalling messages so that it can be delivered to the TCP layer at the CN exactly when the handoff completes. Along with the cross-layer notification regarding the handoff, the TCP sender gets the information regarding the bandwidth and delay of the old and the new access links. The enhancements proposed here are TCP sender-specific and are invoked only upon the arrival of the handoff notification. So in the absence of the handoff notification, we get the performance of the regular TCP.

The enhancement to the TCP sender algorithm (with SACK [16] option enabled) to minimize the unnecessary retransmissions due to packet reordering in a vertical handoff is given in Figure 3. The parameters used in the proposed algorithm such as BDP of an access link and the RTT of the data segment-ACK pair traversing an access link are calculated as follows:

$BDP_{<link>} = BW_{<link>} \cdot RTT_{<link>}$

$RTT_{<link>} = 2 \cdot D_{<link>} + SDpkt_{<link>} + SDack_{<link>}$

where

$SDpkt_{<link>}$ - Data packet serialization delay on the access link

$SDack_{<link>}$ - Serialization delay of ACK on the access link

$D_{<link>}$ - Propagation delay of the access link

$BW_{<link>}$ - Bandwidth of the access link

When handoff notification arrives with the information regarding the old and the new access links

// Congestion possible due to bandwidth/BDP decrease?
If ($FlightSize > 2 \cdot BDP_{newlink}$)
    Set $cwnd\_reduction$ to 1
If (($cwnd\_reduction = 1$) AND
  ($BDP_{oldlink} > BDP_{newlink}$) AND
  ($BW_{newlink} < 8 \cdot BW_{oldlink}$))
    Set *cwnd* and *ssthresh* to $max(2, BDP_{newlink})$
If (TCP is not already in Loss recovery)
  // False fast retransmit possible due to reordering?
  If (($BW_{newlink} > 3 \cdot BW_{oldlink}$) AND
    ($cwnd\_reduction = 0$))
      set $reordering\_flag$ to 1
      dupthresh $= max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$

In Fast retransmit:

  Retransmit the first unACKed segment
  If ($reordering\_flag = 1$)
    Save *cwnd* in $cwnd\_prev$

In Fast Recovery:

  If ($reordering\_flag = 1$)
    Send a new segment for every *dupack*
    If (# of *dupacks* > *dupthresh*)
      Set $return\_fastrecovery$ to 1
      Return to the normal fast recovery

On the arrival of a new ACK indicating that all packets sent before handoff are ACKed:

  Reset $cwnd\_reduction$
  If ((DSACK indicates that the retransmission
    after the handoff was unnecessary) AND
    ($return\_fastrecovery = 0$) AND
    (*cwnd* $< min(cwnd\_prev, BDP_{newlink})$)
      Set *cwnd* to min($cwnd\_prev, BDP_{newlink}$)
      Set *ssthresh* to max($cwnd\_prev, BDP_{newlink}$)
  Reset $reordering\_flag, return\_fastrecovery$
  Reset *dupthresh* to 3
  If (there is a significant change in delay)
      Update the RTT variables

Figure 3: Algorithm to minimize the unnecessary retransmissions due to packet reordering in a vertical handoff

The enhanced TCP algorithm is invoked when a handoff notification arrives with the information regarding the old and the new access links. The algorithm is executed only if there is no imminent congestion in the new path. At the time of a handoff, if the *FlightSize* is greater than the buffering capacity of the new link, packet losses due to congestion may occur after the handoff. Therefore we check whether the *FlightSize* exceeds the estimated buffering capacity of the new link. We assume that the router queue size in front of the access link equals the BDP of the link so that the link has a total buffering capacity of twice the link BDP. If there is congestion, we set the flag *cwnd_reduction*, and modify *cwnd* and *ssthresh* as follows. The *cwnd* and *ssthresh* are set to $BDP_{newlink}$ if the BDP of the old link is greater than the BDP of the new link and the new link bandwidth is less than 8 times the old link bandwidth; otherwise *cwnd* and *ssthresh* remain unchanged. We observe from our experiments that if the new link bandwidth is greater than 8 times the old link bandwidth, the TCP sender is able to recover packet losses due to decrease in BDP using *fast recovery*. Reducing the *cwnd* in this scenario will reduce the sending rate thereby adversely affecting the performance of TCP. Any procedure related to DSACK detection and change of *dupthresh* is invoked only if there is no congestion in the new link at the time of handoff.

The enhanced algorithm makes use of the cross-layer information to determine whether reordering due to the handoff can lead to a false *fast retransmit*. If the ratio of the new bottleneck link bandwidth to the old bottleneck link bandwidth is greater than the *dupthresh*, enough *dupacks* may be generated to trigger a false *fast retransmit*. If this condition arises, the algorithm sets a flag called *reordering-flag* and $max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$ is set as the *dupthresh*. In *fast retransmit*, if the *reordering-flag* is set, TCP retransmits the first unacknowledged segment and saves the *cwnd* in a variable *cwnd_prev*. For every *dupack* which arrives in *fast recovery*, TCP sends a new segment to keep the ACK-clock running until one of the following events occurs: (1) If the total number of *dupacks* exceeds the *dupthresh*, TCP goes back to *fast recovery* and sets the flag *return_fastrecovery*, (2) If all the packets sent before handoff have been acknowledged, TCP resets the *dupthresh* back to the normal value of 3. If the segment retransmitted after the handoff is identified as unnecessary by the DSACK information, and TCP has not returned to *fast recovery*, the algorithm infers that the *dupacks* are generated by packet reordering and are not due to congestion and calculates the new *cwnd* and *ssthresh* values. If the current *cwnd* is less than both *cwnd_prev* and $BDP_{newlink}$, then TCP sets *cwnd* to the smaller of the *cwnd_prev* and the $BDP_{newlink}$ and *ssthresh* to the larger value so that TCP can slow start to find the new path characteristics.

The main advantages of our algorithm are (1) it is conservative in that it will not restore the *cwnd* and *ssthresh* if congestion exists in the network path, (2) it utilizes the new path partially while waiting for the packets to arrive through the old path, and (3) it effectively uses the DSACK information to set the *cwnd* and *ssthresh* of the new path.

4. Simulation results

In this set of experiments, we consider the vertical handoff scenarios where packet reordering occurs, i.e., the handoffs from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link. In order to study the effect of the change in BDP after a handoff, we categorize our experiments into the following three classes: (1) handoff between same BDP links, (2) handoff from a high BDP to a low BDP link, and (3) handoff from a low BDP to a high BDP link.

The simulation environment is the same as that described in our earlier paper [8]. We consider a single TCP flow from the CN to the MN. The TCP packet size is 1500 bytes with the TCP/IP headers included. In our experiments a 20-second interval is chosen to cover all the phases of a TCP connection and a handoff can occur uniformly in any of the 200 points at 100 ms intervals. The duration of each test run includes the completion of the handoff occurring in the 20-second interval. No link errors are modelled as we assume that the packet losses are due to congestion. This choice is justified as the present study is devoted to the effect of vertical handoff on TCP. In this set of experiments, the handoff occurs from a low-bandwidth/high-delay link to a high-bandwidth/low-delay link. The old link bandwidth/delay is fixed at 200 Kbps/300 ms and the new link bandwidth/delay is varied.

To study the behaviour of TCP with vertical handoff we focus on how TCP behaves immediately after the handoff. As a performance index, we calculate the time taken to transfer (to get the acknowledgment) 'n' new data packets after a handoff through the new path where n varies from 50 to 200. We report the results only for the case where n is 100 as the results obtained are similar for all values of n.

4.1. Handoff between same BDP links

In this set of experiments the old link bandwidth/delay is fixed at 200 Kbps/300 ms and the new link bandwidth/delay is varied between 400 Kbps/150 ms and 6400 Kbps/9 ms as shown in Figure 4a. With regular TCP we observe that packet reordering due to a handoff from a high-delay link to a low-delay link triggers false *fast retransmits* in more than 80 % of the handoff points except in the case of a handoff to the 400 Kbps/150 ms link in which not enough *dupacks* are generated. As a result many packets are retransmitted

unnecessarily and *cwnd* and *ssthresh* are reduced. The corresponding results for enhanced TCP show that unnecessary retransmissions are avoided.

Figure 1 compares the behaviour of regular TCP and enhanced TCP for a handoff at 12 sec from 200 Kbps/300 ms to 800 Kbps/75 ms link. We can see that as enhanced TCP retransmits only the first unacknowledged segment, the fast retransmit triggered by the unnecessary retransmissions is avoided and enhanced TCP is able to send more packets than regular TCP which accounts for its slightly better (10 %) performance.

Figure 2 shows the behaviour of regular TCP and enhanced TCP when the handoff occurs at 20 sec from a 200 Kbps/300 ms link to a 1600 Kbps/37 ms link. Even though the time taken for both regular and enhanced TCP to send 100 packets is almost the same, enhanced TCP avoids unnecessary retransmissions and window reduction.

As the ratio of the delay of the old and new link increases (handoff to 6400 Kbps/9ms link) regular TCP is able to send more packets through the new fast link even though there are unnecessary retransmissions and window reduction. Enhanced TCP waits for all the packets through the old slow link to arrive and does not fully utilize the capacity of the new link even though it is transmitting one packet per *dupack* which results in poor performance (20 %). In such situations where the new link has a significantly higher bandwidth and lower delay than the old link, we have to examine whether it is preferable to wait for the packets through the old link so that we can avoid unnecessary retransmissions and *cwnd* reduction or to utilize the available capacity of the high-bandwidth link although a part of the capacity is wasted in unnecessary retransmissions.

4.2. Handoff from a high BDP link to a low BDP link

In this set of experiments the new link BDP is five packets half that of the old link. When there is a BDP decrease after a handoff our algorithm to mitigate the effect of packet reordering is not invoked as the fast retransmit algorithm may be triggered due to packet losses caused by BDP decrease. We can see in Figure 4b that in all handoff scenarios except the case of the handoff to a 400 Kbps/75 ms link enhanced TCP behaves similar to regular TCP. In the case of a handoff to a 400 Kbps/75 ms link, regular TCP needs RTO recovery to recover the lost packets due to BDP change whereas with enhanced TCP the packet losses are kept to a minimum as we set the *cwnd* to the BDP of the new link, yielding about 20 % reduction in transfer time. For the handoff to 800 Kbps/37 ms link, setting the *cwnd* to the BDP of the new link, reduces the packet losses but the reduction in sending rate nullifies the improvement achieved. For the handoffs to 1600Kbps/ 18 ms link, 3200Kbps/9 ms link and 6400 Kbps/ 4 ms link as our algorithm is not at all invoked the behaviour is similar to regular TCP.

4.3. Handoff from a low BDP link to a high BDP link

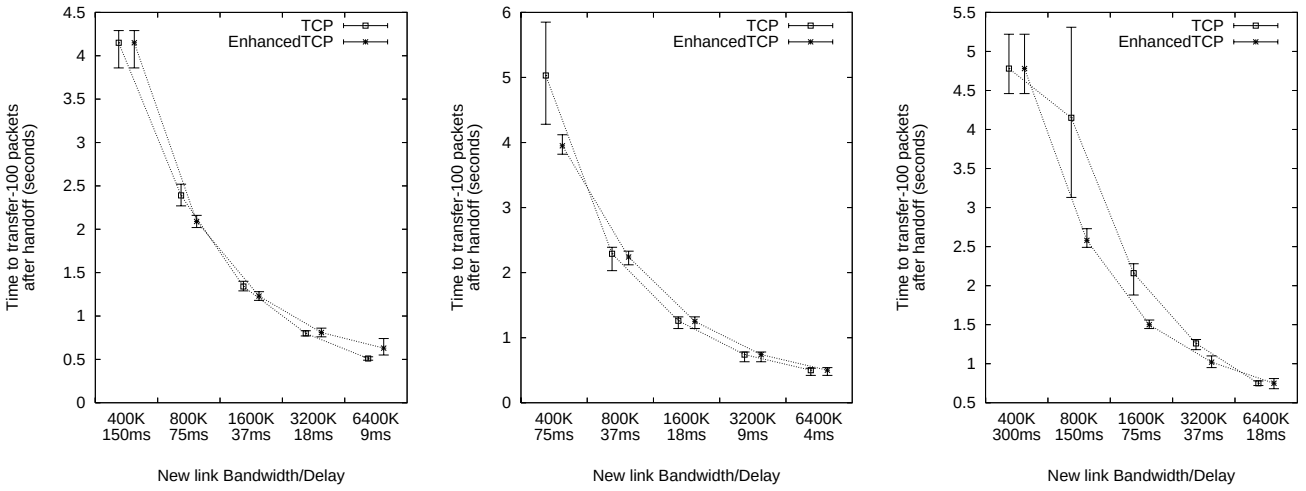
In this set of experiments the new link BDP is 20 packets, double that of the old link. As can be seen in the Figure 4c, the proposed algorithm is effective (up to 30 % reduction in transfer time) with the increase in BDP after a handoff. Our algorithm is not invoked in the handoff to 400 Kbps/ 300 ms link as there is no scenario leading to a false *fast retransmit*. In the case of handoff to 800Kbps/ 150 ms link, regular TCP suffers from multiple fast retransmits similar to the case described in Figure 1a resulting in high variation in the quartile values, whereas enhanced TCP yields a 38 % reduction in transfer time. For the cases of handoff to 3200 Kbps/37 ms and 6400 Kbps/ 18 ms links, enhanced TCP avoids unnecessary retransmissions even though the transfer time is comparable with that of regular TCP. In general, avoiding the unnecessary retransmissions as well as window reduction and slow starting to the new BDP makes the enhanced TCP perform better than regular TCP.

5. Conclusions and future work

In this paper we have proposed an enhancement to the TCP sender algorithm which makes use of the cross-layer notifications to avoid the problems of TCP due to packet reordering in a make-before-break vertical handoff. Simulation results presented in the paper show the effectiveness of the algorithm when the bandwidth(or delay) ratio of the access links varies over a wide range. Further study is required to see when it is preferable to utilize the high capacity link available after a handoff instead of taking measures to avoid unnecessary retransmissions.

As it is difficult as well as unreliable to obtain the exact bandwidth and delay of the access links, we have used the link characteristics available from the MN as hints or bounds for setting the initial values of the TCP congestion control parameters. We need to study how the information obtained by probing the end-to-end path characteristics can be integrated with the cross-layer notification for a faster convergence of TCP parameters after a vertical handoff.

The study on the effect of vertical handoff on TCP and the algorithm proposed are based on our experiments with a single TCP flow. In the presence of multiple flows, the algorithms are likely to require further enhancements. In addition to the experimentation of the algorithm with multiple flows we intend to evaluate the algorithm in real network environments.



(a) To same BDP (b) To low BDP (c) To high BDP

Figure 4: Transfer time for 100 packets after a make-before-break handoff (a) to a same BDP link (b) to a low BDP link (c) to a high BDP link with varying bandwidth/delay of the new link while the old link is fixed at 200Kbps/300ms (200 repetitions). The x-axis shows the link parameters and the y-axis shows the lower quartile, median, and upper quartile of the time (in seconds) to transfer 100 new packets after the handoff.

References

[1] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, Jan. 2001.

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

[3] J. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6):789–798, 1999.

[4] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton. Improving the Robustness of TCP to Non-Congestion Events. IETF RFC 4653, Apr. 2006.

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

[6] E. Blanton and M. Allman. Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions. RFC 3708, Feb. 2004.

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

[8] L. Daniel and M. Kojo. Using Cross-layer Information to Improve TCP performance with Vertical Handoffs. In *Proc. 2nd International Conference on Access Networks (Accessnets 2007)*, Aug. 2007.

[9] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgment (SACK) Option for TCP. RFC 2883, July 2000.

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

[11] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

[12] J. Korhonen, A. Makela, S. Park, and H. Tschofenig. Link and Path Characteristic Information Delivery Analysis. Internet-Draft "draft-korhonen-mobopts-delivery-analysis-01.txt", Oct. 2006. Expired.

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

[14] R. Ludwig and M. Meyer. The Eifel Detection Algorithm for TCP. RFC 3522, Apr. 2003.

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

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, Oct. 1996.

[17] Network Simulator ns-2. ns-2.29, 2005.

[18] S. Park, M. Lee, J. Korhonen, and Z. Zhang. Link Characteristic Information for Mobile IP. Internet-Draft, Jan. 2007. Expired.

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

[20] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol Enhancements for Intermittently Connected Hosts. *ACM Computer Communication Review*, 35(2):5–18, July 2005.

[21] S. Schütz, N. Koutsianas, L. Eggert, W. Eddy, Y. Swami, and K. Le. TCP Response to Lower-Layer Connectivity-Change Indications. Internet-Draft, Feb. 2008. Work in progress.

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