The Performance of Multiple TCP Flows with Vertical Handoff

Laila Daniel
Department of Computer Science
P.O. Box 68, University of Helsinki
FI-00014 Finland
laila.daniel@cs.helsinki.fi

Markku Kojo
Department of Computer Science
P.O. Box 68, University of Helsinki
FI-00014 Finland
markku.kojo@cs.helsinki.fi

ABSTRACT

The performance of an individual TCP flow with a vertical handoff has been studied in several papers. However, the effect of a vertical handoff on *multiple* TCP flows has been little studied. In this paper we study the behaviour of multiple competing TCP flows with a vertical handoff. As a part of this study we evaluate the cross-layer assisted TCP enhancements for a vertical handoff that we had earlier proposed and analyzed for a single TCP flow. We show that our algorithms can be adapted for multiple TCP flows with minor modifications and that they are effective in improving multiple flow-TCP performance in the presence of a vertical handoff.

Categories and Subject Descriptors

C.2.1 [**N**]: etwork Architecture and Design; C.2.2 [**N**]: etwork Protocols; C.4 [**P**]: erformance of Systems

General Terms

Performance, Algorithms

Keywords

Vertical Handoff, TCP, Wireless Access Networks, Cross-layer notifications

1. INTRODUCTION

The problem of Transmission Control Protocol (TCP) [25] behaviour in the presence of vertical handoffs [22] has grown in significance with the proliferation of wireless access networks to connect to the Internet and has been an active research area in recent years [8, 9, 12–15, 26, 31]. These studies have shown that the potentially significant differences in the bandwidth and/or delay of the two access links involved in a vertical handoff can affect TCP performance. The major problems of TCP due to a vertical handoff are the unnecessary retransmissions and congestion window (*cwnd*) reduction due to spurious RTOs and packet reordering as well as packet losses due to abrupt changes in the link capacity and link disconnection. Several cross-layer assisted enhancements have been proposed in the literature to mitigate these problems and to improve TCP performance [7–9, 12–15, 18–20, 26, 29, 30, 32]. These studies have focused on the effect of handoff on a *single* TCP flow. However, the effect of competing TCP flows has not been taken into account in evaluating the various proposed solutions. As multiple parallel TCP flows tend to affect TCP behaviour in general and more so in a vertical handoff, the resulting change in TCP behaviour has to be studied and taken into account in the proposed solutions. Such an approach in evaluating the adaptations of TCP algorithms in a dynamic environment has been suggested previously in [2] as a step closer to real-world networks.

In this paper we first study how TCP performance is affected by a vertical handoff when multiple TCP flows are present and then describe how the cross-layer assisted TCP enhancements for vertical handoff that we had proposed earlier for a single TCP flow [6,9] can be easily adapted for the case of multiple TCP flows. We show that with the inclusion of the number of simultaneous TCP flows in the cross-layer information, we can easily modify our earlier algorithms to adapt to multiple flow scenarios.

The rest of the paper is organized as follows. Section 2 gives a overview of the related research work on methods to improve TCP performance with a vertical handoff. Section 3 describes our algorithms to adapt TCP to a vertical handoff when multiple TCP flows are present. Section 4 describes the results of the simulation experiments to evaluate these algorithms. Section 5 presents the conclusions of the study.

2. RELATED WORK

We present an overview of the research dealing with the problems of TCP in the presence of vertical handoffs. The solutions proposed in these papers are in the context of a single TCP flow. To the best of our knowledge the behaviour of multiple TCP flows in a vertical handoff has not been described explicitly in the research literature. We categorize the related work based on the problems of vertical handoff they try to solve.

To avoid the problem of spurious retransmission timeouts (RTOs) that may occur due to a vertical handoff, [19] suggests that upon receiving a handoff notification, a TCP receiver should send the acknowledgement for a received packet through both the old and new interfaces and also reset the RTO value to 3 seconds. The use of ICMP messages to calculate the new RTO value after a handoff has been proposed in [26] as a solution to the spurious RTO problem. The TCP-Eifel detection algorithm [21] uses the TCP timestamps option [5] to detect spurious RTOs. Forward-Retransmission Timeout (F-RTO) algorithm [28] is a TCP sender-only algorithm that helps to detect spurious RTOs.

Setting the *cwnd* appropriately after a handoff is crucial both in avoiding the congestion-related losses due to a handoff to a lower

MobiWac'09, October 26–27, 2009, Tenerife, Canary Islands, Spain.

bandwidth-delay product (BDP) link as well as in effectively utilizing the higher BDP of a new link after a handoff. Slow-starting to find the new *cwnd* value after a handoff is proposed in [18]. The use of the Quick-Start algorithm [27] to find the correct sending rate with the help of routers is proposed in [29]. In the proposals [20, 32] the BDP of the end-to-end path after a handoff is calculated as the product of the bottleneck link capacity estimated using the packet-pair algorithm [17] and the round-trip time (RTT) and then the *cwnd* and slow-start threshold (*ssthresh*) are set to this BDP value. An explicit *cwnd* reduction trigger from the mobile node which signals the difference between the BDP values of the old and new link is proposed in [14]. *BDP probing*, proposed in [19], is a technique which initially sends two back-to-back packets just after a handoff to estimate the capacity of the bottleneck link and then sends the data packets at the rate of the estimated capacity to find the available bandwidth. A receiver-based mechanism proposed in [12] addresses the problem of abrupt change in link capacity due to a vertical handoff. When an impending handoff is detected, the TCP receiver sends the receiver advertised window (*rwnd*) based on the BDP of the new network and this *rwnd* will be effective once the mobility registration is complete. Overbuffering is suggested in [13] to reduce the effects of BDP change due to a handoff.

To mitigate the problems arising from packet reordering due to a vertical handoff, a *nodupack* scheme is proposed in [14] and it suppresses the transmission of *dupacks* during a handoff. However, this may delay the loss recovery if the reordering occurs due to a handoff. An RTT-equalization scheme is proposed in [26] which sends the acknowledgements for the packets received from the fast interface through the slow interface and vice versa. DSACK [11], an extension of SACK in which the receiver reports to the sender the receipt of a duplicate segment, can be used to detect packet reordering and to undo the consequent congestion control actions [3].

To minimize the unused connection time after a handoff, a TCP retransmission trigger that causes TCP to attempt a retransmission when the connectivity is restored after a handoff is proposed in [30].

A detailed account of the research work in this area can be found in Chapter 5 of [6].

3. TCP ENHANCEMENTS FOR MULTIPLE FLOWS

In this section we present a brief overview of the problems of TCP in vertical handoffs and describe the modifications we propose in this paper to our earlier algorithms [6,7,9] in order to adapt them for handoffs in the context of multiple TCP flows. Our algorithms make use of the cross-layer information regarding the occurrence of a handoff, the bandwidth and delay of the access links involved in the handoff and the number of TCP flows undergoing the handoff. Our modelling assumption is that the mobile node can send the above information to the TCP sender at the correspondent node along with the mobility signalling.

3.1 Spurious RTOs

Spurious RTOs occur when a make-before-break handoff [22] occurs from a low-latency link to a high-latency link. After the handoff, the acknowledgments (ACKs) will be delayed due to the higher delay of the new link. TCP retransmission timer expires before the arrival of the ACKs through the new link due to the small RTO value calculated on the basis of the old path. This spurious RTO will cause unnecessary retransmission of packets and reduction in *cwnd* and *ssthresh*. Unnecessary retransmissions waste the bandwidth and reduction in *cwnd* and *ssthresh* reduce the sending rate, resulting in performance degradation.

In order to avoid the occurrence of spurious RTOs we calculate the minimum RTO (*minrto*) based on the new access link delay and update the RTO timer immediately so that the new *minrto* comes into effect. As a result any change in the delay of the end-to-end path will be reflected better in the RTO calculation. As the *minrto* calculation is based on the access link delay alone, the RTT variables are initialized as in RFC 2988 upon the arrival of the ACK for the data sent through the new access link. This enables the RTO to adapt to the end-to-end RTT quickly. No modification to this algorithm is needed for multiple TCP flow scenarios. Due to space limitations this algorithm is not given here and the reader is referred to [6, 9] for details.

3.2 Packet Losses due to Congestion

Packet losses can occur when there is a decrease in BDP after a handoff. When a handoff occurs from a high-bandwidth link to a low-bandwidth link, significant packet losses can occur even when the BDP of the two links remains the same.

We first introduce the algorithm used to avoid the packet losses due to a decrease in BDP in the case of a single TCP flow and then describe the modification to this algorithm for the case of multiple TCP flows which is shown in Figure 1. We set the *cwnd* and *ssthresh* to the BDP of the new access link if the *FlightSize* at the time of handoff is greater than twice the BDP of the new access link (or 1.5 times the BDP of the new access link in case the bandwidths of the old and new link differ significantly). A detailed discussion of the rationale of this algorithm can be found in [6,9]. When there are multiple flows sharing a bottleneck access link it is no longer appropriate to set the *cwnd* of each TCP flow to the BDP of the new link. Accordingly the *cwnd* and *ssthresh*) is set to the BDP of the new link divided by the number of concurrent flows that use the link at the time of handoff. This value corresponds to a single flow's share of the bandwidth when a number of flows share the bottleneck link. Here we make the assumption is that the bottleneck link is the last/first hop wireless access link. This assumption is justifiable as the bandwidth of the wired links in an end-to-end path is usually much higher than that of the wireless access links at its end points.

When a handoff notification arrives
If ((TCP not in RTO recovery)
If ($BW_{oldlink} >= 8 * BW_{newlink}$)
If (FlightSize $> 1.5 * (\mathbf{BDP_{newlink}/N})$)
/* N refers to the number of flows */
$cwnd\_reduction = 1$
Else if ($BW_{oldlink} < 8 * BW_{newlink}$)
If (FlightSize $> 2 * (\mathbf{BDP_{newlink}/N})$
$cwnd\_reduction = 1$
If ($cwnd\_reduction == 1$)
cwnd $= \mathbf{max}(2, \mathbf{BDP_{newlink}/N})$
$ssthresh = cwnd$
$cwnd\_reduced = 1$

Figure 1: Algorithm to reduce congestion-related packet losses for multiple flows. Adaptation to multiple flows shown in bold.

Figures 2 and 3 describe our algorithms to reduce the unused connection time and to combat the problems arising from packet reordering in a vertical handoff in the setting of multiple TCP flows. In these algorithms also we set the *cwnd* and *ssthresh* in the same manner based on the number of simultaneous TCP flows through

the bottleneck link. This is the only modification necessary to adapt our algorithms designed for a single TCP flow to the case of multiple TCP flows in the presence of a vertical handoff. As the information about the number of flows can be easily included in the handoff notifications, our algorithms are easy to implement in practice.

3.3 Unused Connection Time and ssthresh Reduction

When a handoff notification arrives:
If (TCP in RTO recovery)
Retransmit the first unacknowledged packet
Set *ssthresh* to $\max(\mathbf{2}, \mathbf{BDP_{newlink}/N})$
/* N refers to the number of flows */
If there is a significant change in delay
Initialize RTT variables as for a new connection
When ACK for new data arrives
Update RTT variables

Figure 2: Algorithm to reduce the unused connection time and to set *ssthresh*. Adaptation to multiple flows shown in bold.

Figure 2 gives our algorithm to reduce the unused connection time and to set the *ssthresh* in a break-before-make handoff. When a break-before-make handoff occurs, the end-to-end path between the mobile node and correspondent node is broken and the connectivity resumes only after the handoff is completed resulting in packet losses. If the disconnection period is greater than the current RTO value, the retransmission timer expires. The TCP sender retransmits the first unacknowledged segment and doubles the RTO value. For each subsequent timer expiration for the same segment, TCP doubles the RTO value again [24]. When the end-to-end connection is up, the TCP sender needs to wait until the retransmission timer expires again before attempting another retransmission. This unused connection time can be up to one minute [24] depending on the disconnection length and the next scheduled RTO and it increases the recovery time of the lost packets. If more than one timeout has occurred, i.e., a retransmission is considered to be lost, the *ssthresh* value is further reduced in some implementations resulting in inefficient recovery. By retransmitting the first unacknowledged segment immediately if the TCP sender is in RTO recovery when the handoff notification arrives and setting the *ssthresh* to the BDP of the new link scaled by the number of flows, our algorithm given in Figure 2 is able to mitigate the problems due to a long disconnection in a break-before-make handoff.

3.4 Packet Reordering

Packet reordering is a problem that TCP faces when there is a significant reduction in delay after a make-before-break handoff. The packets sent through a low-delay link after the handoff may overtake the packets transmitted through the high-delay link before the handoff and this causes packet reordering which generates *dupacks*. If the TCP sender receives a *dupthresh* number of *dupacks* (typically 3) it enters fast recovery, halves the *ssthresh* and *cwnd* and continues in congestion avoidance. The retransmission and the reduction in *ssthresh* and *cwnd* are unnecessary as the *dupacks* which arrive are due to packet reordering and not due to congestion.

We briefly describe the main idea behind the algorithm given in Figure 3 and refer the reader to [7] for its detailed description. If the bandwidth of the new access link is greater than *dupthresh* times the bandwidth of the old access link, there is a possibility of packet reordering leading to false fast retransmit. If this condition arises, a new *dupthresh* value is calculated based on the ratio of the bandwidth of the two access links. In fast retransmit, the TCP sender saves the previous *cwnd* value if there is a possibility of reordering. In fast recovery, the TCP sender sends a new segment for every arriving *dupack* until all the segments transmitted before the handoff are acknowledged or the number of *dupacks* exceeds the *dupthresh*. In the latter case TCP returns to the normal fast recovery [10]. If the retransmission is identified as unnecessary using DSACK information, the *cwnd* and *ssthresh* are set to the BDP of the new access link scaled by the number of flows.

When a handoff notification arrives with the information regarding the old and the new access links
/* Congestion likely due to bandwidth or BDP decrease ? */
If ($\mathbf{FlightSize} > \mathbf{2} \cdot (\mathbf{BDP_{newlink}/N})$)
Set $cwnd\_reduction$ to 1
If (($cwnd\_reduction = 1$) AND
($BDP_{oldlink} > BDP_{newlink}$) AND
($BW_{newlink} < 8 \cdot BW_{oldlink}$))
Set *cwnd* and *ssthresh* to $\max(\mathbf{2}, (\mathbf{BDP_{newlink}/N}))$
/* N refers to the number of flows */
If (TCP is not already in Loss recovery)
/* False fast retransmit likely due to reordering ? */
If (($BW_{newlink} > 3 \cdot BW_{oldlink}$) AND
($cwnd\_reduction = 0$))
set $reordering\_flag$ to 1
dupthresh $= max(\frac{BW_{newlink}}{BW_{oldlink}}, 3)$
In Fast retransmit:
Retransmit the first unACKed segment
If ($reordering\_flag = 1$)
Save *cwnd* in $cwnd\_prev$
In Fast Recovery:
If ($reordering\_flag = 1$)
Send a new segment for every *dupack*
If (number of *dupacks* $>$ *dupthresh*)
Set $return\_fastrecovery$ to 1
Return to the normal fast recovery
On the arrival of a new ACK indicating that all packets sent before handoff are ACKed:
Reset $cwnd\_reduction$
If ((DSACK indicates that the retransmission after the handoff was unnecessary) AND
($return\_fastrecovery = 0$) **AND**
($\mathbf{cwnd} < \mathbf{min}(\mathbf{cwnd\_prev}, (\mathbf{BDP_{newlink}})/\mathbf{N})$)
Set *cwnd* to min(cwnd_prev, $(\mathbf{BDP_{newlink}/N})$
Set *ssthresh* to max(cwnd_prev, $(\mathbf{BDP_{newlink}/N})$
Reset $reordering\_flag, return\_fastrecovery$
Reset *dupthresh* to 3
If (there is a significant change in delay)
Update the RTT variables

Figure 3: Algorithm to combat the problems arising from packet reordering in a vertical handoff. Adaptation to multiple flows shown in bold.

3.5 Slow RTO convergence

After a handoff, the RTO will converge to the RTT of the new path very slowly. One reason for this is that the formula for updating the smoothed RTT (SRTT) value at the TCP sender gives a much smaller weight to the current RTT sample compared to that of the previous SRTT value. Another reason is that the RTT variables are updated only once in an RTT and not for each ACK received [24]. In order to quickly converge to the RTO value corre-

sponding to the new path, we initialize the RTT variables as in RFC 2988 if the old and the new RTT values differ by a factor of two or more and the new RTT value is greater than *minrto*. No modification to this algorithm is needed for multiple TCP flow scenarios. Due to space limitations we do not give this algorithm here and the reader refer to to [6, 9].

4. SIMULATION RESULTS

We use the ns-2 network simulator to model the behaviour of TCP in a vertical handoff. This section describes the simulation experiments and our results obtained. We first discuss the simulation setup, and then move on to investigating different types of vertical handoff scenarios.

4.1 Simulation Setup

In the simulation model the mobile node is capable of using both the access links involved in a vertical handoff. Both access links have dedicated base stations that are connected to a common wireless access router by 100 Mbps links. The router has a 100 Mbps connection to a server (correspondent node) in the fixed network. The propagation delay over each of the fixed links is 2 ms. Unless otherwise stated, the router buffer size of each link is set to $max(BDP_{<link>}, 5)$ packets. We consider bulk TCP flows from the correspondent node to the mobile node. A detailed description of the simulation environment is given in [6, 9].

The baseline TCP used in the experiments is TCP SACK [4] and is referred to as regular TCP. TCP SACK with the enhancements we had proposed earlier [6, 9] is referred to as Enhanced-TCPv0 (ETCPv0) and the TCP SACK in conjunction with the algorithms described in Section 3 is referred to as Enhanced-TCPv1 (ETCPv1). The TCP packet size is 1500 bytes including the TCP/IP headers.

In our experiments a 20-second interval is chosen to cover all the phases of a TCP connection and a handoff can occur uniformly in any one of the 200 instances at 100 ms intervals. The duration of each test run includes the completion of the handoff occurring in the 20-second interval. No link errors are modelled as we assume that the packet losses are solely due to congestion.

In order to study the behaviour of TCP with vertical handoff we focus on the TCP behaviour immediately after a handoff. As a performance metric, we calculate the time taken to transfer (to receive the acknowledgment) a specific number of packets. This time is calculated with respect to the slowest flow. In comparing the performance of the enhanced TCP with the regular TCP, we use the median value of the time taken to transfer 100 packets after a handoff. In all the performance graphs given in this paper, the x-axis shows the number of flows and the y-axis shows the lower quartile, median, and upper quartile of the time (in seconds) to transfer 100 packets after the handoff.

We are categorizing our experiments into two classes, (i) handoff from a fast link to a slow link and (ii) handoff from a slow link to a fast link. We have chosen the following four sets of bandwidth and delay combinations to reflect the situations arising in handoff involving access networks such as EGPRS [23], HSDPA [1], WiMAX [33] and WLAN [16]. A rough range of the bandwidth and propagation delay (one-way) of the access networks such as EGPRS (200 Kbps/300 ms), HSDPA (700 Kbps/75 ms, 2000 Kbps/50 ms, 6000 Kbps/50 ms), WiMax (2000 Kbps/50 ms, 11000 Kbps/20 ms) and WLAN (11000 Kbps/10 ms, 54000 Kbps/2 ms) is used in our experiments. In the first two experiments, the BDP of the access links held constant while the BDP of the access links in the second experiment is higher. In the second class of experiments the BDP of the two access links involved in a handoff differ. Here we perform two sets of experiments in which the BDP of the access links in the second set is higher than that in the first set. In each of the experiments we study the behaviour of TCP flows for the case of one, two and four long, simultaneous TCP flows.

4.2 Handoff from a Fast link to a Slow link

In this section we describe a set of experiments where a handoff occurs from a fast link to a slow link. As the problems that occur with a make-before-break handoff differ from that of a break-before-make handoff we discuss the two cases separately.

4.2.1 Make-Before-Break Handoff

The main problems of TCP in a make-before-break handoff from a fast link to a slow link are the occurrence of spurious RTOs and unnecessary retransmissions associated with them in addition to the packet drops due to a decrease in bandwidth. Our experiments described here show that the algorithms designed to avoid the spurious RTOs and to reduce the packet losses that are described in Section 3 are effective and improve the performance of TCP.

First we consider a handoff occurring between same BDP links even though the bandwidth and delay of the new access link differs considerably from that of the old link. The BDP of both access links is 10 packets. The problems in this case are the occurrence spurious RTOs and unnecessary retransmissions associated with them in addition to the packet drops due to a decrease in bandwidth.

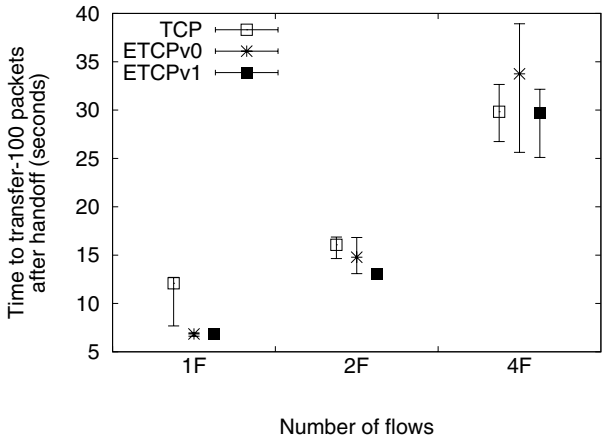

Figure 4: Time taken to transfer 100 packets after a make-before-break handoff from a 6400Kbps/9ms link to a 200Kbps/300 ms link

Figure 4 shows the time taken by regular TCP, ETCPv0 and ETCPv1 to transfer 100 packets after a make-before-break handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link. For a single TCP flow ETCPv1 and ETCPv0 show a 40 % reduction in transfer time compared to that of regular TCP. In the case of two flows a similar reduction in transfer time is seen whereas in the case of four flows the performance of regular TCP and ETCPv1 are nearly the same. The reason for this is that with the increase in the number of flows there is a consequent decrease in the size of the *cwnd* of each flow. As a result the typical problems of TCP in vertical handoff due to spurious RTOs and *cwnd* reduction do not have a significant impact on TCP performance. In the case of four flows, we can see that ETCPv0 performs worse than both regular TCP and ETCPv1 as ETCPv0 sets the *cwnd* to the BDP of the new link resulting in an aggressive behaviour leading to packet losses.

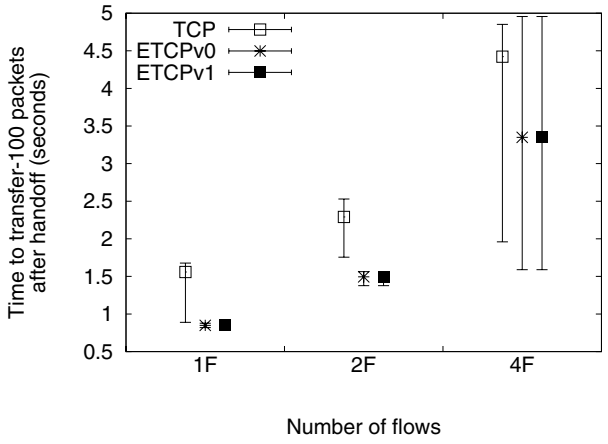

Figure 5: Time taken to transfer 100 packets after a make-before-break handoff from a 54000Kbps/2ms link to a 2000Kbps/50 ms link

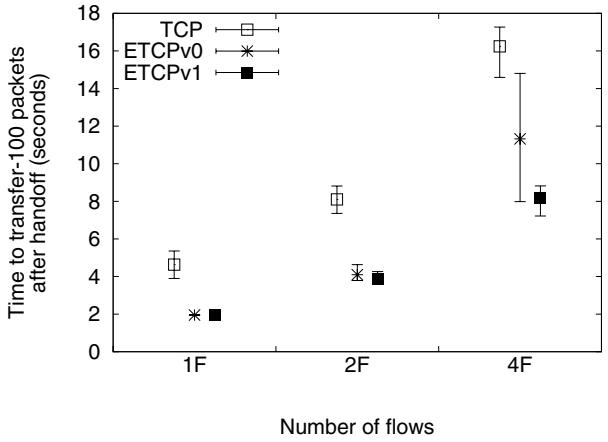

Figure 6: Time taken to transfer 100 packets after a make before-break handoff from a 11000Kbps/10ms link to a 700Kbps/75ms link

In this category of same BDP access link handoffs we next describe a handoff from a 54000 Kbps/2 ms link to a 2000 Kbps/50 ms link. Here the BDP of the links is set to a higher value (18 packets) and a higher bandwidth and lower delay compared to the corresponding values in experiment described above. Figure 5 shows that in the case of a single flow ETCPv1 and ETCPv0 reduce the transfer time for 100 packets after a handoff by about 40 % compared to regular TCP. We can also see from this figure that for two and four flows ETCPv1 still shows up to 30 % improvement in transfer time over regular TCP. With the increase in BDP, the *cwnd* for each flow increases resulting in packet losses that occur due to a make-before-break handoff. ETCPv1 improves the performance over regular TCP as it avoids packet losses due to spurious RTOs and *cwnd* reduction.

In the second class of experiments, the BDP of the access links differ. We have two sets of experiments in this class, namely, a handoff from a 11000 Kbps/10 ms link to a 700 Kbps/75 ms link (BDP of 18 and 9 packets respectively) and a handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link (BDP of 50 and 38 packets respectively).

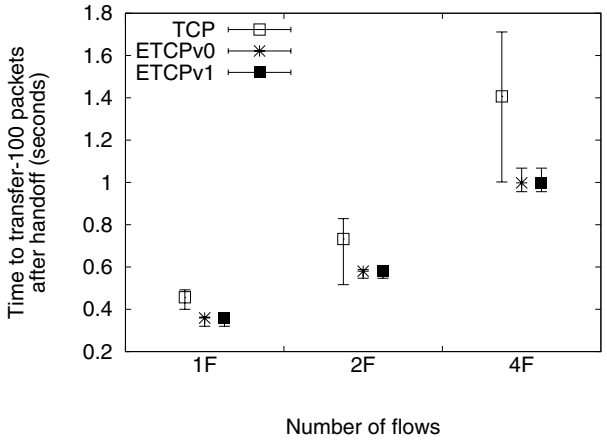

Figure 7: Time taken to transfer 100 packets after a make-before-break handoff from a 54000Kbps/4ms link to a 6000Kbps/50ms link

The main problem of regular TCP in a make-before-break handoff from a 11000 Kbps/10 ms link to a 700 Kbps/75 ms link is the large number of packet losses due to the reduction in BDP (BDP reduction from 18 packets to 9 packets) and also due to large decrease (about 15 times) in bandwidth. RTO recovery is needed to recover the lost packets and more losses may occur before regular TCP adapts itself to the *cwnd* of the new path. On the other hand, ETCPv1 sets the *cwnd* based on the BDP of the new link and the number of flows and is able to avoid the packet losses. Figure 6 illustrates the transfer time taken for regular TCP and ETCPs in this scenario. In the case of a single TCP flow, with ETCPv1 and ETCPv0, there is approximately 60 % reduction in transfer time of regular TCP.

When there are two TCP flows, packet losses lead to RTO recovery in the case of regular TCP but there are no packet losses for ETCPv1. The reduction in transfer time for ETCPv1 is around 50 % as the available bandwidth for a single flow is halved. When the number of flows increases to four, the available bandwidth share of the flows decreases. Regular TCP suffers packets losses and needs RTO recovery, while with ETCPv1 there are no packet losses as it sets the *cwnd* and *ssthresh* to the new link BDP scaled by the number of flows. ETCPv1 is able to reduce the transfer time by about 50 % compared to regular TCP. Figure 6 shows that ETCPv0 reduces the transfer time of regular TCP by about 35 % but there are still packet losses as it sets the *cwnd* and *ssthresh* to the new link BDP which is larger than the BDP corresponding to the flow's share of bandwidth when there are four simultaneous flows.

In a make-before-break handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link, both the access links have sufficiently high BDP values of 50 packets and 38 packets respectively. Figure 7 shows that ETCPv1 shows a 20-40 % reduction in the transfer time compared to the regular TCP even when there are four simultaneous TCP flows. This significant improvement in performance shows the effectiveness of our algorithms when the *cwnd* is sufficiently large.

4.2.2 Break-Before-Make Handoff

Next we describe the experiments involving a break-before-make handoff from a fast link to a slow link. Here we carry out the same set of experiments as in the case of make-before-break handoff described earlier.

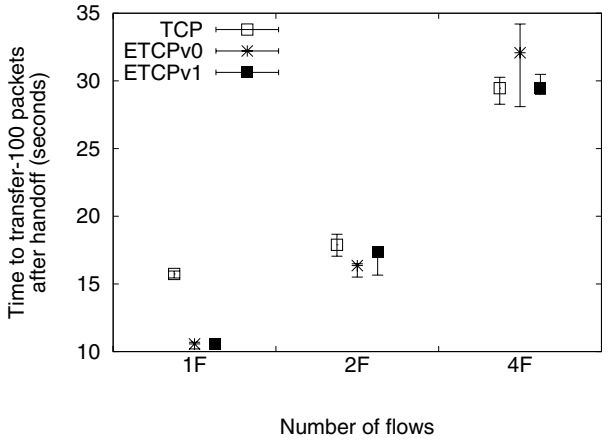

Figure 8: Time taken to transfer 100 packets after a break-before-make handoff from a 6400Kbps/9ms link to a 200Kbps/300ms link

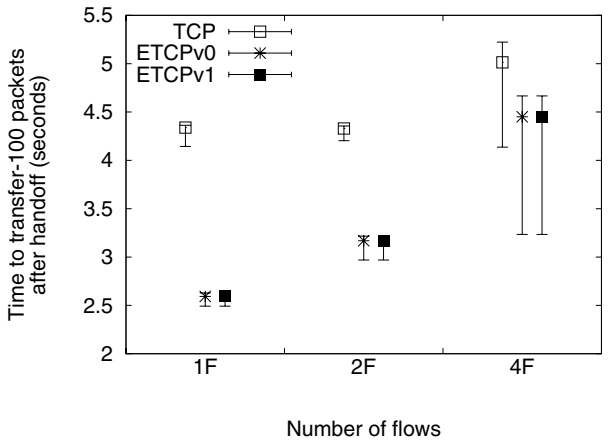

Figure 9: Time taken to transfer 100 packets after a break-before-make handoff from a 54000 Kbps/2ms link to a 2000 Kbps/50 ms link, disconnection period 1s

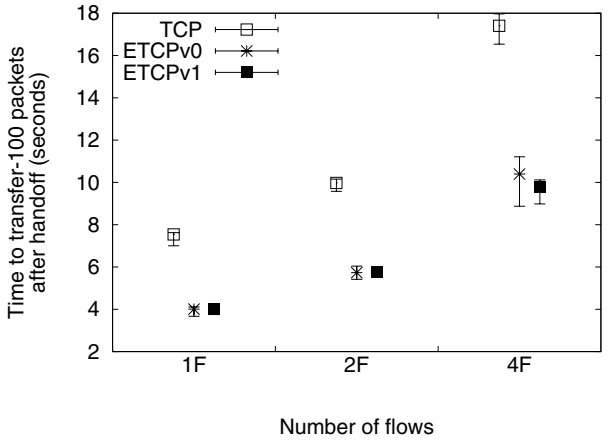

Figure 10: Time taken to transfer 100 packets after a break-before-make handoff from a 11000 Kbps/10ms link to a 700 Kbps/75 ms link, disconnection period 1s

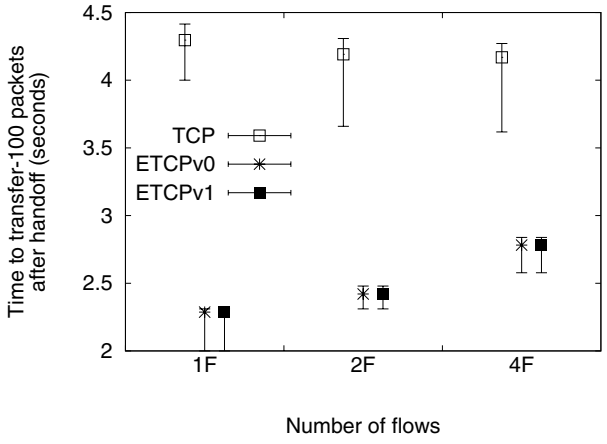

Figure 11: Time taken to transfer 100 packets after a break-before-make handoff from a 54000 Kbps/4ms link to a 6000 Kbps/50 ms link, disconnection period 1s

A break-before-make handoff results in packet losses and unused connection time. The algorithm in Figure 2 retransmits the first unacknowledged packet immediately if TCP is already in RTO recovery when a handoff notification arrives and this helps to utilize the connection as soon as the new access link is up after a handoff. The proper setting of the *ssthresh* by the algorithm helps in avoiding unnecessary reduction of *ssthresh* due to repeated timeouts.

Figure 8 shows the time taken for transferring 100 packets by the three TCP versions in a break-before-make handoff from a 6400 Kbps/9 ms link to a 200 Kbps/300 ms link. We can see that in the case of a single flow both ETCPv0 and ETCPv1 reduce the transfer time by about 40 % compared to regular TCP. As the number of flows increases to four, the *cwnd* available for a single flow decreases and this reduces the number of packet losses due to a disconnection . In the case of four flows, we observe that ETCPv0 incurs additional losses resulting in increased transfer time compared to regular TCP and ETCPv1.

The marked improvement in the performance of the ETCPv1 over the regular TCP in a break-before-make handoff from a 54000 Kbps/2 ms link to a 2000 Kbps/50 ms link can be clearly seen from the Figure 9. Figure 10 shows that for all the flows, ETCPv1 reduces the transfer time by about 50 % compared to regular TCP. Figure 11 shows that ETCPv1 is effective in a break-before-make handoff from a 54000 Kbps/4 ms link to a 6000 Kbps/50 ms link. As the *cwnd* is sufficiently large ETCPv1 performs better than regular TCP when there are four simultaneous TCP flows. We can also see from this figure that there is about 20-40 % reduction in transfer time with ETCPv1 compared to regular TCP for all the flows.

4.3 Handoff from a Slow link to a Fast link

In this section we describe a set of experiments where both make-before-break and break-before-make handoffs occur from a slow link to a fast link.

4.3.1 Make-Before-Break Handoff

Packet reordering is the main problem of TCP in a make-before-break handoff from a slow link to a fast link. After a handoff, packets through the fast new link may arrive at the receiver sooner than the packets sent before the handoff through the slow old link resulting in packet reordering. Our algorithm given in Figure 3 is designed to mitigate the problems arising from packet reordering.

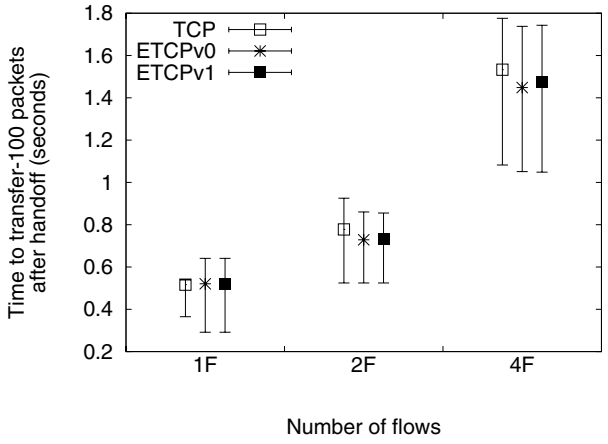

Figure 12: Time taken to transfer 100 packets after a make-before-break handoff from a 200Kbps/300ms link to a 6400Kbps/9ms link

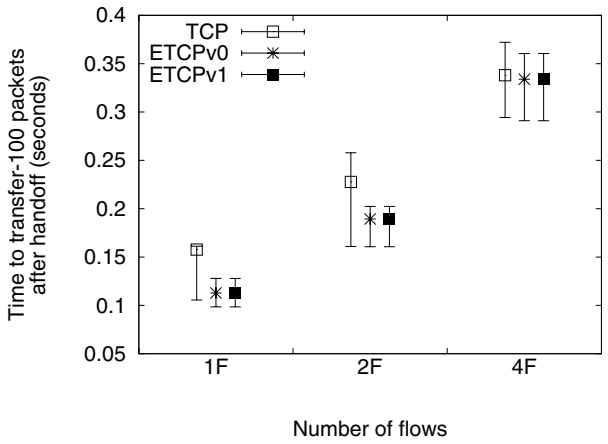

Figure 13: Time taken to transfer 100 packets after a make-before-break handoff from a 2000Kbps/50ms link to a 54000Kbps/2ms link

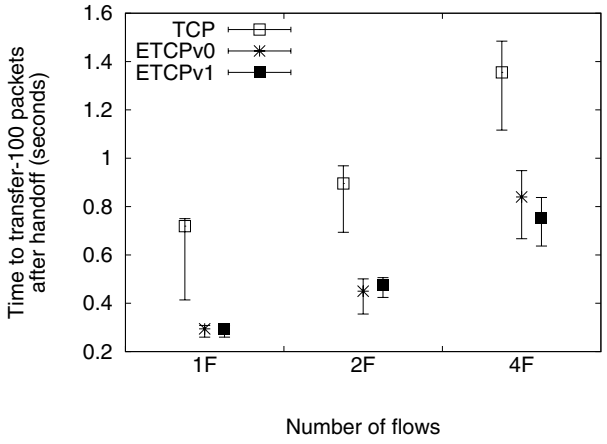

Figure 14: Time taken to transfer 100 packets after make-before-break handoff from a 700Kbps/75ms link to a 11000Kbps/10ms link

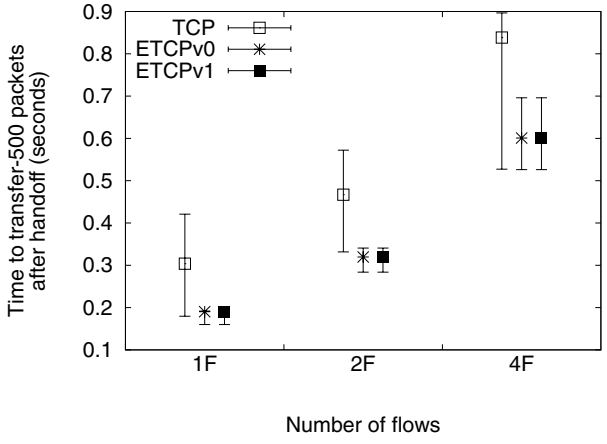

Figure 15: Time taken to transfer 500 packets after make-before-break handoff from a 6000Kbps/50ms link to a 54000Kbps/4ms link

Figure 12 shows the results of a make-before-break handoff from a 200 Kbps/300 ms link to a 6400 Kbps/9 ms link. Even though the transfer time of ETCPv1 and TCP are quite the same, we have observed in our experiments that ETCPv1 reduces the unnecessary retransmissions caused by packet reordering.

Figure 13 shows the results for a make-before-break handoff from a 2000 Kbps/50 ms link to a 54000 Kbps/2 ms link. In the cases of one and two TCP flows our algorithm in Figure 3 reduces the transfer time taken by regular TCP by about 30 %. In the case of four flows, the *cwnd* for a single flow is relatively small and the unnecessary retransmissions and *cwnd* reduction due packet reordering have only a minor effect on TCP performance. We can see from Figure 13 that when there are four simultaneous TCP flows sharing the link at the time of handoff, ETCPv1 and regular TCP have comparable performance. In our experiments we observed that ETCPv1 is effective in reducing the unnecessary retransmissions caused by packet reordering.

Figure 14 shows the results of the transfer time taken by regular TCP and ETCPv1 in a handoff from a 700 Kbps/75 ms link to a 11000 Kbps/10 ms link. Here the handoff is from a low BDP link to a high BDP link. ETCPv1 shows an improved performance, more than 50 % reduction in transfer time of regular TCP for all the flows.

Fig 15 shows the time taken to transfer 500 packets after a make-before-break handoff from a 6000 Kbps/50 ms link to a 54000 Kbps/9 ms link. Packet reordering is again the main problem of TCP in this scenario. ETCPv1 utilizes the new high bandwidth link effectively and is able to transfer 300-400 packets while waiting for the packets sent earlier through the old link. Here we have taken the time to transfer 500 packets after a handoff which is an adequate time in this scenario for TCP to recover from the effects of a handoff. ETCPv1 is able to reduce the transfer time of regular TCP by about 30 % in this scenario.

4.3.2 Break-Before-Make Handoff

In an break-before-make handoff, our algorithm in Figure 2 immediately retransmits the lost segment and thereby helps to utilize the fast link after the handoff.

As shown in Figure 16 for a break-before-make handoff from a 200 Kbps/300 ms link to a 6400 Kbps/9 ms link the transfer time

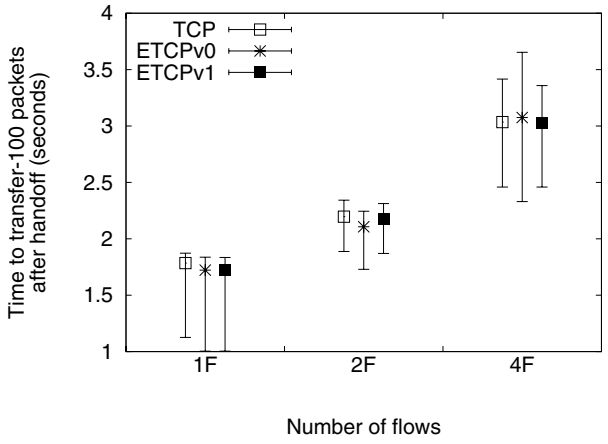

Figure 16: Time taken to transfer 100 packets after a break-before-make handoff from a 200Kbps/300ms link to a 6400Kbps/9mslink, disconnection period 1s

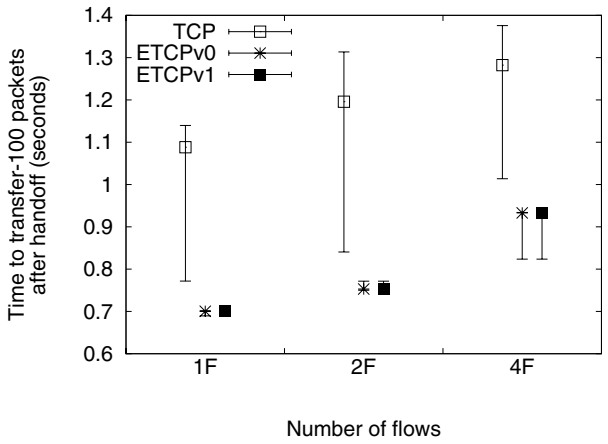

Figure 17: Time taken to transfer 100 packets after a break-before-make handoff from a 2000Kbps/50ms link to a 54000 Kbps/2ms link, disconnection period 1s

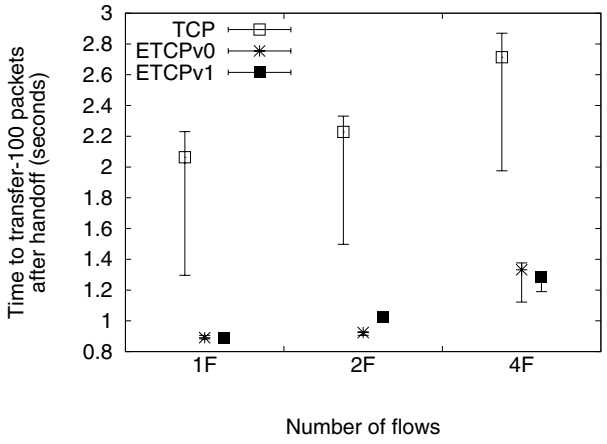

Figure 18: Time taken to transfer 100 packets after a break-before-make handoff from a 700Kbps/75ms link to a 11000Kbps/10ms link, disconnection period 1s

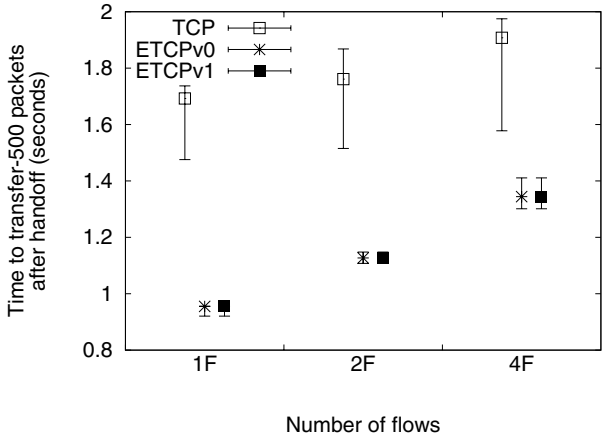

Figure 19: Time taken to transfer 500 packets after a break-before-make handoff from a 6000Kbps/50ms link to a 54000Kbps/4ms link, disconnection period 1s

of ETCPv1 and regular TCP are equal for all the flows. This is because the RTO value of the old access link (2.5 seconds to 3 seconds) is larger than the disconnection period of one second and a timeout will not occur during the disconnection time. Therefore ETCPv1 and ETCPv0 will not enter the algorithm given in Figure 2. When a disconnection period is four seconds or longer the algorithm in Figure 2 will be entered and consequently there will be significant improvement arising from its use.

We can observe from Figure 17 that ETCPv1 shows nearly 50 % reduction in transfer time over regular TCP when a break-before-make handoff occurs from a 2000 Kbps/50 ms link to a 54000 Kbps/2 ms link. The reason for this improvement is due to ability of ETCPv1 to utilize the high bandwidth link immediately after a handoff by using the algorithm in Figure 2 whereas regular TCP waits for the next RTO to start the transmission after the handoff. We can see from Figure 18 that in a handoff from a 700 Kbps/75 ms link to a 11000 Kbps/10 ms link ETCPv1 shows 50 % reduction in transfer time over regular TCP for all the flows.

Figure 19 shows the time taken for transferring 500 packets after a break-before-make handoff from a 6000 Kbps/50 ms link to a 54000 Kbps/4 ms link. We can see that ETCPv1 obtains over 50 % reduction in transfer time over regular TCP for all the flows we consider in our experiments. The reason for the improvement is the same as given in the previous paragraph.

5. CONCLUSIONS

In this paper we study the behaviour of multiple TCP flows in the presence of a vertical handoff. Through extensive simulations we show that the proposed cross-layer assisted algorithms, which utilize the information about the number of simultaneous TCP flows and the bandwidth and delay of the access links, are effective in avoiding the problems of TCP due to a vertical handoff and improve TCP performance. The problems of TCP in a vertical handoff due to the number of unnecessary retransmissions and packet losses are aggravated with the increase in the size of the *cwnd*. With the increase in the number of TCP flows the size of the *cwnd* decreases roughly in inverse proportion to the number of TCP flows that share the bottleneck access link. Consequently, as the number of simultaneous TCP flows increases, the typical problems of TCP due to spurious RTOs, packet reordering and *cwnd* reduction

arising from a vertical handoff tend to have diminishing impact on TCP performance, in particular if the *cwnd* is small. However, if the *cwnd* is sufficiently large the algorithms proposed in this paper will be effective for multiple TCP flows in various vertical handoff scenarios.

Acknowledgments

This work has been carried out in the frame of the WISEciti research project. We would like to thank our colleagues in the Wireless Internet group and the Wiseciti consortium for discussions and support in carrying out this research.

6. REFERENCES

[1] 3GPP. High Speed Downlink Packet Access (HSDPA); Overall UTRAN description. Technical Report 3GPP TS 25.855, Mar. 2002.

[2] M. Allman and A. Falk. On the Effective Evaluation of TCP. *ACM Computer Communication Review*, 5(29):59 – 70, Oct. 1999.

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

[4] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP. IETF RFC 3517, Apr. 2003.

[5] D. Borman, R. Braden, and V. Jacobson. TCP Extensions for High Performance. IETF RFC 1323, May 1992.

[6] L. Daniel. TCP Performance with Vertical Handoff. Licentiate Thesis, Series of Publications C, No. C-2008-221, Nov. 2008. Available at: http://www.cs.helsinki.fi/u/ldaniel/lic-thesis-tcp-vho.pdf.

[7] L. Daniel, I. Järvinen, and M. Kojo. Combating Packet Reordering in Vertical Handoff Using Cross-Layer Notifications to TCP. In *Proc. IEEE Conference on Wireless and Mobile Computing, (WiMob08)*, Oct. 2008.

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

[9] L. Daniel and M. Kojo. Employing cross-layer assisted tcp algorithms to improve tcp performance with vertical handoffs. *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, 1(4/5/6):433–465, 2008.

[10] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 3782, Apr. 2004.

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

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

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

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

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

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

[17] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of ACM SIGCOMM '91*, pages 3–15, 1991.

[18] S.-E. Kim and J. A. Copeland. TCP for Seamless Vertical Handoff in Hybrid Mobile Data Networks. In *Proc. IEEE Globecom 2003*. IEEE, Dec. 2003.

[19] D. Li, K. Sleurs, E. V. Lil, and A. V. de Capelle. A fast adaptation mechanism for TCP vertical handover. In *Proc. of the International Conference on Advanced Technologies for Communications, ATC 2008*, Oct. 2008.

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

[21] R. Ludwig and R. H. Katz. The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communication Review*, 30(1):30 – 36, Jan. 2000.

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

[23] D. Molkdar, W. Featherstone, and S. Lambotharan. An Overview of EGPRS: the packet data component of EDGE. *Electronics and Communication engineering Journal*, 14:21–38, Feb. 2002.

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

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

[26] H. Rutagemwa, S. Pack, X. Shen, and J. W. Mark. Robust cross-layer design of wireless-profiled tcp mobile receiver for vertical handover. *IEEE Transactions on Vehicular Technology*, 56(6):3899–3911, Nov. 2007.

[27] P. Sarolahti, M. Allman, and S. Floyd. Determining an Appropriate Sending Rate Over an Underutilized Network Path. *Computer Networks (Elsevier)*, pages 1815–1832, May 2007.

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

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

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

[31] M. Stemm and R. H. Katz. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

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

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