Effect of Vertical Handovers on Performance of TCP-Friendly Rate Control

Andrei Gurtov
andrei.gurtov@teliasonera.com

Jouni Korhonen
jouni.korhonen@teliasonera.com

TeliaSonera Finland

An intersystem or vertical handover is a key enabling mechanism for next generations of mobile communication systems. A vertical handover can cause an abrupt change of up to two orders of magnitude in link bandwidth and latency. It is hard for end-to-end congestion control to adapt promptly to such changes. This is especially a concern for slowly responsive congestion control algorithms, such as TCP-Friendly Rate Control (TFRC). TFRC is designed to provide a smooth transmission rate for real-time applications and, therefore, is less responsive to changes in network conditions than TCP. Using measurements and simulation, we show that TFRC has significant difficulties adapting after a vertical handover. TFRC receives only a fraction of TCP throughput over a fast link, but can be grossly unfair to concurrent TCP flows after handover to a slow link. We show that two proposals based on overbuffering and an explicit handover notification are effective solutions to these problems. Using them, TFRC can quickly adapt to new link characteristics after a handover, while otherwise maintaining a smooth transmission rate.

I. Introduction

Existing wireless networks offer to a mobile user a trade-off between connection bandwidth, coverage, and cost. The user can utilize the most suitable wireless network at a given time and location, for example, by switching between Wireless LAN (WLAN), General Packet Radio Service (GPRS), and Universal Mobile Telecommunications System (UMTS) links while keeping ongoing data transfers. An intersystem handover is also known as a *vertical handover* because wireless networks often form an overlay structure — the fastest network with least coverage is contained in a slower network with a larger coverage area [38].

An intersystem handover is challenging to end-to-end transport protocols, because packets often get lost, delayed or reordered during a handover. Furthermore, path characteristics such as bandwidth, latency, and the buffer size can change instantly, often more than by an order of magnitude. Estimators used by the end-to-end transport protocols to control the amount of outstanding data in the network and the rate of transmission are likely to be significantly off after a handover. As a result, overshooting or underutilization of the available bandwidth becomes likely.

In the Internet, TCP is the dominant transport protocol that serves well many applications requiring reliable data delivery. However, for real-time applications, such as streaming video, a highly variable transmission rate of TCP is problematic. Recently, several slowly responsive congestion control algorithms were proposed based on the notion of TCP-friendliness [32, 43]. Such algorithms provide a smooth transmission rate on the short time scale. On the longer time scale they consume no more bandwidth than a TCP flow under similar network conditions. The TCP-friendly Rate Control (TFRC) [11] is perhaps the most popular protocol among proposed alternatives.

In this paper, we evaluate performance of TFRC during handovers between GPRS, WLAN, and UMTS. We measure behavior of TFRC and TCP flows in a testbed implementing vertical handovers using Mobile IP. To verify our testbed measurements and to study the effect of changes in path characteristics, we use an *ideal handover* model in the ns-2 simulator [40]. Essentially, an ideal handover is represented by a step change in the bottleneck link bandwidth, latency, and buffer size, as if a smooth handover with packet forwarding were implemented [7]. Throughput, aggressiveness, responsiveness, and fairness of TFRC are evaluated. We show that there are significant problems with using TFRC in the presence of vertical handovers. In particular, over a fast link TFRC receives only a fraction of TCP throughput, while over a slow link TFRC can starve concurrent TCP flows after a handover. Two proposals based on overbuffering and an explicit handover notification are demonstrated to be effective solutions to these problems.

The rest of the paper is organized as follows. Sec-

tion II gives the necessary background on end-to-end transport protocols and congestion control. In Section III, we describe our Mobile IP testbed and present TFRC and TCP traces from vertical handovers. In Section IV, behavior of TFRC and TCP during an ideal handover is explored via simulation. In Section V, we examine the effect of TFRC parameters. In Section VI and VII, we introduce and evaluate overbuffering and the explicit handover notification for improving aggressiveness and responsiveness of TFRC and TCP. Finally, Section VIII presents main conclusions from this work.

II. Background and Related Work

In this section, we review congestion control mechanisms in TCP and TFRC, as well as existing work on evaluating the effect of vertical handovers on transport protocols.

II.A. End-to-end Congestion Control

II.A.1. Transmission Control Protocol

TCP is a reliable transport protocol that uses slow start and congestion avoidance for congestion control [3]. TCP has an important property of self-clocking, also known as the packet conservation principle [26]. Assuming that delayed acknowledgments are not used, in the equilibrium condition every arriving ACK indicates that a segment has left the network and triggers a transmission of a new segment. We assume that the reader is familiar with the TCP protocol [39] and only review existing work on TCP performance in the presence of handovers.

A study on the effect of mobility on TCP found that packet losses during a handover significantly reduce throughput [30]. The most significant factor contributing to long TCP recovery from handovers was found to be in the exponential back-off of the TCP retransmit timer [8]. A proposed solution is to artificially generate three Duplicate ACKs at the TCP receiver to trigger fast retransmit at the sender and avoid lengthy recovery using the retransmit timer. However, this approach may not work if Selective Acknowledgments (SACK) are used by the TCP connection. An improved variant of this mechanism is proposed by Fladenmuller and Silva [10]. As an alternative to modifying TCP, Hsieh proposed a new receiver-centric transport protocol that performs well in the presence of handovers [21].

The transmission rate of transport protocols using a sliding window is defined by the rate of returning acknowledgments. The number of outstanding segments — the current window — depends on the buffer size of the bottleneck link, that is typically set according to the bandwidth-delay product of the link. As an example, TCP congestion control estimators (the congestion window and slow start threshold) for two links (1000 kbps/10 ms and 100 kbps/100 ms) are the same assuming the same link buffer size. Therefore, after a handover between two such links, the TCP sender instantly adapts to the bandwidth of a new link. In other words, window-based protocols, such as TCP, are more sensitive to the change of the bandwidth-delay product of the link than only of the link bandwidth.

TCP options negotiated at the connection establishment may not be appropriate after a handover to the network with vastly different characteristics. Unfortunately, TCP options cannot be adjusted later in the connection lifetime. We identified option values that are adequate for all overlay networks considered in this paper. These values are listed in Appendix A.

II.A.2. TCP-Friendly Rate Control

TFRC permits an application to transmit at a steady rate that is typically within a factor of two from the TCP rate in the similar conditions [11]. TFRC does not halve the transmission rate after a single packet loss, but is also slow to increase the rate in the absence of congestion. In other words, the main goal of TFRC is to provide a smooth transmission rate, but not to aggressively make use of available bandwidth. In the absence of explicit feedback from the network, there is an inherent trade-off between smoothness of the transmission rate and convergence time to the fair share of bandwidth.

The TFRC receiver reports the loss event rate p and the average receive rate X_{rcv} to the sender. The sender computes the reference transmission rate X_{calc} based on p, X_{rcv}, and average round-trip time using a TCP rate equation [32]. The actual transmission rate X is set as follows [20]:

$if(p > 0)$
  $X_{calc} = equation\ rate$
  $X = max(min(X_{calc}, 2 * X_{recv}), s/t_{mbi})$
$else$
  $if(t_{now} - tld >= R)$
    $X = max(min(2 * X, 2 * X_{recv}), s/R)$
  $tld = t_{now}$

Here s represents the packet size, tld is time when the rate was last doubled, R is RTT, and t_{mbi} represents the maximum back-off time (64 seconds by de-

fault) in the persistent absence of feedback. If p is zero, no packet loss has yet been seen by the flow. In this phase, the TFRC sender emulates slow start of TCP by doubling the transmission rate every RTT.

TCP does not typically reduce the congestion window more than once per a window of data. Therefore, calculating the loss event rate rather than simply taking the packet loss rate is an important part of TFRC. The default method that TFRC uses for calculating the loss event rate is called the Average Loss Interval. With this method a weighted average of recent intervals between packet losses is computed. The weights are 1, 1, 1, 1, 0.8, 0.6, 0.4, and 0.2 for the oldest loss interval.

History discounting allows the TFRC receiver to adjust the weights, concentrating more on the most recent loss interval, when it is more than twice as large as the computed average loss interval. This is an optional mechanism to allow TFRC to response somewhat more quickly to the sudden absence of congestion, as represented by a long current loss interval.

Self-clocking is seen as the key feature of TCP congestion control that contributes to the stability of the Internet [5]. An optional self-clocking mechanism for TFRC is applied for the RTT following a packet loss. It limits the sender's rate to at most the received rate in the previous round trip time. Furthermore, in the absence of losses, the TFRC maximum sending rate is limited to the earlier receive rate times a constant to prevent a rapid increase in the transmission rate.

Main metrics of a congestion control algorithm are throughput, fairness, aggressiveness, responsiveness, and smoothness. Throughput is the rate at which data is delivered to the receiver. Fairness reflects the ability of a flow to share bandwidth in a compatible way with a TCP flow running in similar conditions. Aggressiveness describes how rapidly the algorithm increases the transmission rate in the absence of congestion. Responsiveness reflects how fast the rate is decreased in time of persistent congestion. Finally, smoothness defines how variable is the rate when packet losses are relatively rare. Formally, the responsiveness of a congestion control mechanism has been defined as the number of round-trip times of persistent congestion until the sender halves its sending rate, where persistent congestion is defined as the loss of one packet per round-trip time [11]. The aggressiveness of a congestion control mechanism has been defined as the maximum increase in the sending rate in one round-trip time, in packets per second, given the absence of congestion [5].

The maximum increase of TFRC rate given fixed RTT is estimated to be 0.14 packets per RTT and 0.22 packets per RTT with history discounting [11]. It takes four to eight RTTs for TFRC to halve its sending rate in the presence of persistent congestion.

We explained in Section II.A.1 that window-based protocols, such as TCP, are sensitive to changes in the delay-bandwidth product, but not necessarily to changes in bandwidth. For rate-based protocols, such as TFRC, the opposite is true. TFRC does not estimate the amount of outstanding data necessary to utilize the link, but transmits at a relatively steady rate. Therefore, TFRC is more sensitive to changes in the link bandwidth than in the delay-bandwidth product.

TFRC is not a full-fledged transport protocol, as it only concerns with end-to-end congestion control. Therefore, TFRC should be deployed together with a transport protocol, such as UDP, RTP, or Datagram Congestion Control Protocol (DCCP) [27].

In this paper, we examine aggressiveness and responsiveness of TFRC during step changes in link characteristics triggered by a vertical handover. Fairness and smoothness are considered only briefly. Our study goes further than previous work [45, 11, 5] in several ways. First, we evaluate changes in link bandwidth and latency of up to two orders of magnitude. Second, we consider the effect of varying RTT. Third, we are interested in cellular networks where little degree of statistical multiplexing is present. This allows us to concentrate on behavior of one or two flows. Our results are based on measurements and simulation. We are not aware of other TFRC measurements over wireless links except by Beaufort et al. [6].

II.B. Overlay Networks

The terms wireless overlay networks and a vertical handover were introduced during the Bay Area Research Wireless Access Network project [38]. The BARWAN testbed included WaveLAN, Infrared, Ricochet wireless networks, and later a wide-area cellular network [42]. Other researchers built a number of similar testbeds, concentrating on minimizing delays and packet losses during handovers.

Several studies evaluated performance of a Mobile IP [34] handover in overlay networks. A common conclusion appears to be that while Mobile IP can provide sufficiently quick handovers for non-real-time applications, the disruption is too long to be tolerated by real-time applications [10]. However, using optimizations, handover times as low as 10 ms can be achieved in WLANs [9]. Local loss recovery using a snoop proxy was shown to improve TCP performance during handovers [4]. A study of an optimized smooth

Table 1: Link characteristics of overlay networks.

System	RTT, ms	Bw, Mbps	Bw*RTT, Kbytes	Coverage
GPRS	600	0.03	∼2	country
UMTS	300	0.384	∼15	city
WLAN	10	11	∼14	building
LAN	1	100	∼13	desk

handover in Mobile IP observed that forwarding packets from the old access point to the new one can significantly reduce packet losses [7, 33].

Hsieh and Seneviratne compared several mechanisms for improving performance of Mobile IP for TCP [22]. The basic Mobile IPv6 framework is compared with Hierarchical Mobile IPv6, Hierarchical or Flat Mobile IPv6 with Fast handover, Simultaneous Bindings, and Seamless handoff architecture for Mobile IP (S-MIP) for linear and ping-pong mobility scenarios. All frameworks except S-MIP suffered from packet losses and performance degradation.

In this paper, we consider four different overlay networks: GPRS, UMTS, WLAN, and LAN. These network technologies are described, for example, by Walke [41]. Their link characteristics are summarized in Table 1. Vertical handovers cause a varying degree of change in link characteristics. GPRS-UMTS handovers trigger a change in the delay-bandwidth product, while WLAN-LAN handovers trigger a significant change in bandwidth. The delay-bandwidth product of WLAN and LAN is similar.

III. Measurements of Vertical Handovers

In this section, we describe our testbed and present measurement results of TFRC flows during vertical handovers. For comparison, we also show traces of a TCP flow running alone and concurrently with TFRC.

III.A. Measurement Setup

Figure 1 shows the network architecture that we use for measurements. Mobile nodes can connect to the testbed using 100 Mbps Ethernet LAN, 11 Mbps 802.11b WLAN, a live GPRS network, and a live UMTS network. For brevity, we only present measurements results of handovers between GPRS and WLAN.

The connection to the GPRS and UMTS cellular networks is realized using a dedicated Access Point Name (APN). IP traffic from GPRS and UMTS is sent over a Generic Router Encapsulation (GRE) tunnel

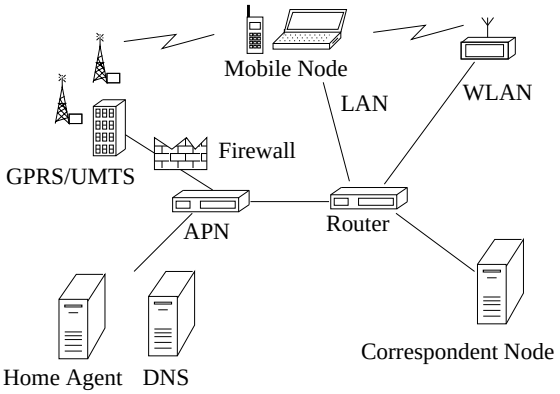


Figure 1: Measurement testbed based on Mobile IP.

between a Gateway GPRS Support Node (GGSN) and the APN router. This is necessary because the firewall in the live cellular network would otherwise drop Mobile IP messages. The Mobile Node, Correspondent Node, Home Agent, and the APN router are PCs with a Pentium-3 600 MHz processor running the Linux operating system. The APN router has a Debian distribution with a 2.2.17 kernel. The Mobile Node, Home Agent, and Correspondent Node have the RedHat 7.3 distribution.

For GPRS access we used a Nokia's D211 PCMCIA card, which is capable of three downlink and one uplink timeslots. With CS-2 coding it can achieve 36 kbps downlink and 12 kbps uplink transfer speeds. Our testbed has a commercial SecGo Mobile IPv4 installation. This Mobile IPv4 implementation is based on the previous work done in the Dynamics research project [12]. The SecGo Mobile IPv4 implementation is fully compliant to the latest specification [34] and also implements NAT Traversal (NATT) tunneling [28]. The Mobile IPv4 product we used does not implement any handover enhancements, such as a smooth handoff [7]. Buffering in the Home Agent and in the Mobile Node does not modify standard Linux buffering.

In handover tests we used a co-located Foreign Agent residing at the Mobile Node. We forced reverse NATT tunneling and defined zero agent solicitations to be sent from the Mobile Node. These settings caused all traffic to go through the Home Agent. During a handover, the Mobile Node sends a registration request message immediately to the Home Agent using a new link. The new and old links are simultaneously active, the layer two handover delay is zero and all delay is at the layer three. Handovers were manually forced by changing the interface prioritization from the client software graphical interface. It may not be a practical scenario for vertical mobility,

Table 2: Delay and packet loss during handovers as measured in the testbed.

From → to	Delay, seconds	DL loss, %	UL loss, %
GPRS→ LAN	13	100	-
LAN→ GPRS	3	0	100
GPRS→ WLAN	1	95	-
WLAN→ GPRS	4	0	100
LAN→ WLAN	0.2	0	100
WLAN→ LAN	0.8	0	100

but is sufficient for our purposes of studying the effect of a change in link characteristics on end-to-end congestion control.

NATT tunneling adds header overhead of 32 bytes, which consists of 20 bytes of encapsulating IP-header, 8 bytes of UDP headers, and 4 bytes of NATT tunneling header. The IP Maximum Transfer Unit (MTU) was intentionally lowered to 1440 bytes to avoid packet fragmentation. We concentrated on downlink bulk data transfers from the server to the mobile node. TCP traffic is generated with `ttcp`, and TFRC traffic with an application-level implementation of TFRC over UDP [23]. This implementation does not completely correspond to the TFRC specification, but it was the best implementation available. Packet traces were recorded at the end hosts using `tcpdump`.

III.B. TCP Measurement Results

Table 2 summarizes the delay and packet loss estimated from TCP traces during handovers in our testbed. No packet duplication or reordering was observed during the measurements. The delay refers to the total duration of the handover on the IP layer and does not include, for example, the effect of TCP timeouts. The fraction of lost packets during handovers is given separately for downlink (DL) and uplink (UL) directions. During handovers from WLAN and LAN, packets were rarely lost in downlink, but always in the uplink direction.

During handovers from GPRS, all packets in downlink were lost and we did not have sufficient information to estimate losses in uplink. TCP transmits acknowledgments only when data segments are arriving. As we only experimented with downlink transfers, loss of data segments in the downlink direction stops transmission of acknowledgments in uplink. Without any packets sent in the uplink direction, it is not possible to estimate the loss probability for uplink traffic.

In the rest of this section, we focus on traces of TCP connections during vertical handovers between GPRS and WLAN. Handovers to and from LAN have had a similar pattern and we do not include those traces here. Figure 2(a) shows TCP behavior during a handover from GPRS to WLAN. The graph shows a time-sequence trace of TCP segment numbers modulo 90 from the sender side. The handover takes a second to execute. Although almost all data segments were lost, a single acknowledgment arriving after the handover resumes the connection quickly. Linux TCP uses the FACK algorithm [36] that enables triggering a fast retransmit after a single Duplicate ACK rather than waiting for three Duplicate ACKs as required by the standard TCP [3]. In experiments where the TCP sender had to rely on the retransmit timeout to recover lost segments, we observed connection breaks of more than ten seconds during handovers from GPRS to WLAN and LAN. The reason for such breaks is a high latency and queuing delay in GPRS.

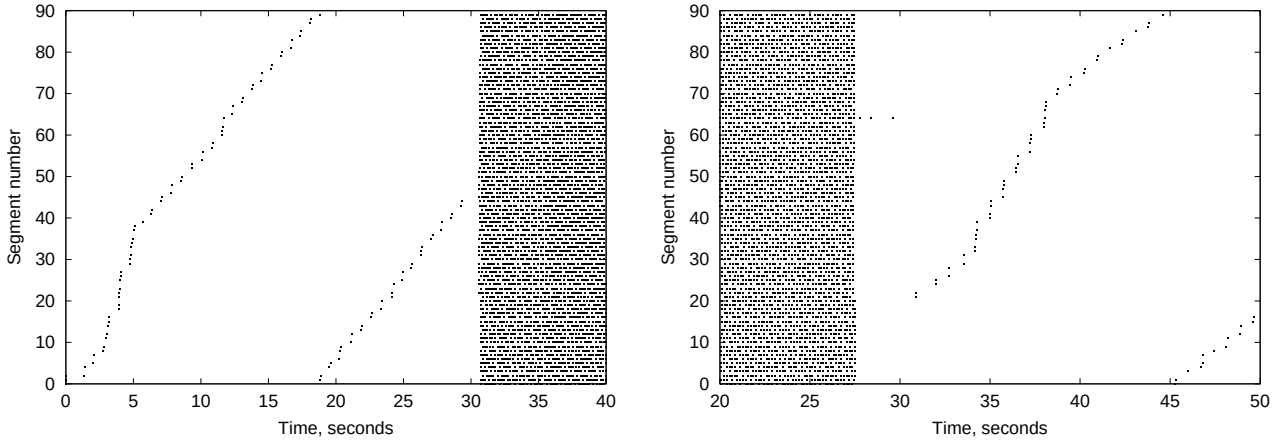
A handover from WLAN to GPRS in Figure 2(b) lasted four seconds. The TCP sender timed out and performed three retransmissions using exponential back-off. Interestingly, the first acknowledgment that returns to the sender after a handover on the 31st second confirms all outstanding segments. This tells us two things. First, all data segments outstanding when the handover has started were delivered to the receiver. Second, all acknowledgments except the one for the highest outstanding segment were lost. TCP generates an acknowledgment for at least every second segment; if they arrived to the sender we would see unnecessary go-back-N retransmissions [29].

III.C. TFRC Measurement Results

Figure 3(a) and 3(b) show behavior of a single TFRC flow during handovers between GPRS and WLAN. As before, the time-sequence trace is recorded at the sender side and the sequence numbers in the graph wrap after 90 segments. Handovers are triggered approximately on the 30th second.

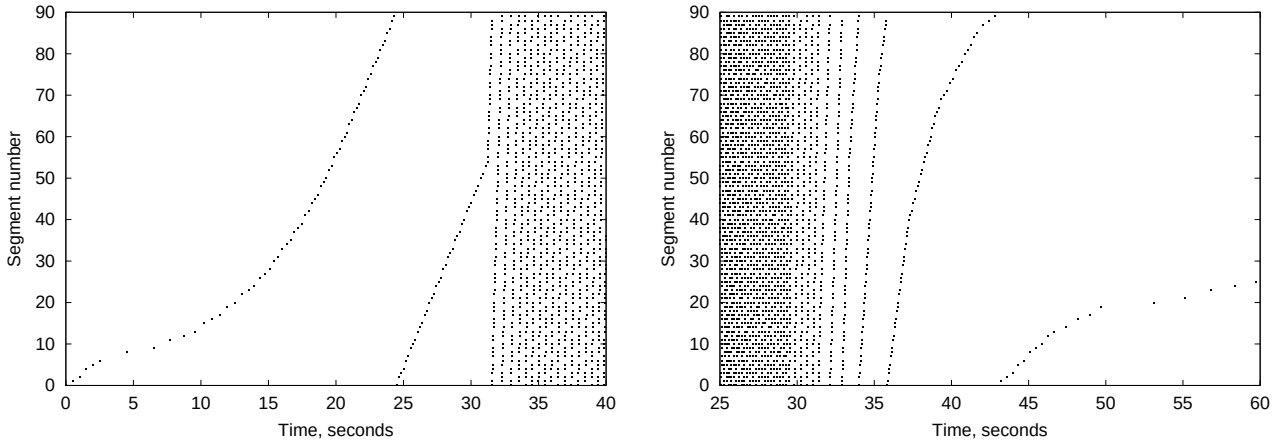
TFRC aggressiveness can be evaluated from Figure 3(a), when a handover is made from a slow (GPRS) to a fast link (WLAN). In this test, the TFRC flow accelerates quickly to the bandwidth of the new link. A possible reason is that the flow has not yet exited the slow start phase when the handover occurs [17]. In slow start, TFRC is much more aggressive than after reaching the steady state with a smooth transmission rate. Then, the past history of high RTT and low bandwidth can make TFRC adaptation slow after a handover from GPRS to WLAN.

The responsiveness of TFRC to a decrease in band-



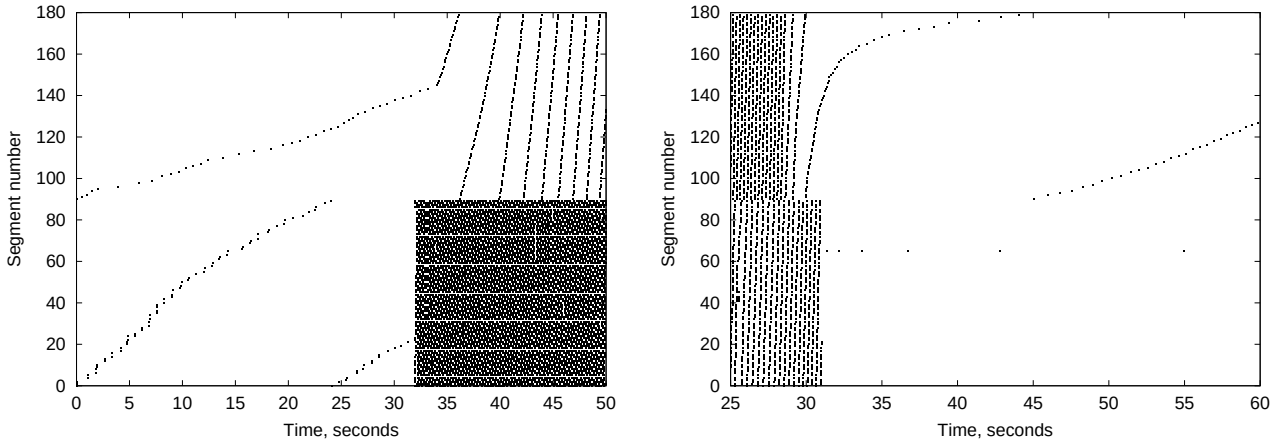
(a) GPRS → WLAN (b) WLAN → GPRS

Figure 2: Measured behavior of a TCP flow during a vertical handover in the testbed (the handover occurs approximately at time 30).



(a) GPRS → WLAN (b) WLAN → GPRS

Figure 3: Measured behavior of a TFRC flow during a vertical handover in the testbed (the handover occurs at time 30).



(a) GPRS → WLAN (b) WLAN → GPRS

Figure 4: Measured behavior of a TFRC (top) and TCP flow (bottom) during a vertical handover in the testbed (the handover occurs at time 30).

width after a handover from WLAN to GPRS can be seen in Figure 3(b). On the 30th second feedback packets stop arriving to the sender. About a second later, a no-feedback timer expires at the sender and the transmission rate is halved several times until the first feedback packet arrives on the 40th second. Because TFRC is a rate-based protocol, there is no visible break in transmission during a handover as with TCP. There is a significant time period before the TFRC flow converges to the bandwidth on the new link.

III.D. TFRC Measurement Results with a Competing TCP Flow

In this section, we describe packet traces of a TFRC flow with a competing TCP flow during vertical handovers between GPRS and WLAN. In Figure 4(a) and 4(b), handovers are triggered approximately on the 30th second. Figures show a time-sequence graph from the sender side of TCP and TFRC. Segment numbers modulo 90 are plotted for TCP in the lower and for TFRC in the upper part of the graph.

Figure 4(a) shows TCP and TFRC behavior during a handover from GPRS to WLAN. Prior to handover, both flows are running at similar rates. However, the TFRC flow accelerates slower than the TCP flow after a handover. The TFRC flow obtains only 15% of the throughput of a competing TCP flow. This scenario illustrates that TFRC discovers the increased bandwidth significantly slower than TCP. Several minutes can be needed for TFRC to achieve a fair share of bandwidth after a handover from a slow to a fast link.

Figure 4(b) shows TCP and TFRC behavior during a handover from WLAN to GPRS. As in the previous graph, flows share the bandwidth fairly before the handover. However, after a handover the TCP flow is starved, performing retransmissions using the exponential back-off. Only when the TFRC flow terminates (not shown in the graph) the TCP flow is able to resume transmission. A possible explanation for this behavior is that the TFRC implementation used for measurements has the minimum sending rate of one packet per RTT [11]. On a slow GPRS link, a TFRC flow transmitting at this minimum rate can starve a TCP flow that uses the exponential back-off up to 64 seconds between retransmission attempts. Although the latest TFRC specification [20] requires a similar type of behavior, TFRC still reacts considerably slower than TCP to decreased bandwidth, causing congestion and a high packet loss rate for concurrent flows after a handover from a fast to a slow link.

IV. Simulating Ideal Vertical Handovers

In this section, we evaluate the effect of abrupt changes in link bandwidth, latency, and buffer size on TCP and TFRC flows after a vertical handover.

IV.A. Simulation Setup

In this section, we want to focus on fundamental effects of a change in link characteristics, but not on transient disruptions caused by imperfect handover mechanisms. Therefore, a simple approach presenting a handover as a step change in the bottleneck link bandwidth, latency, and the buffer size is sufficient for our purposes. We model an ideal handover using the ns-2 simulator [40]. We implemented an algorithm described in Appendix B to prevent packet reordering during a handover. The implementation of the Drop-Tail queue was enhanced to check for buffer overflow when the limit of queue size changes.

The network topology is a simple dumbbell, with a mobile node adjacent to the wireless access link (see e.g., [18]). Traffic is generated by uni-directional downlink transfers. A handover is triggered on the 30th second after the start of simulation. The bottleneck queue is Drop-Tail. The bandwidth and one-way latency of the link are set according to Table 1. The end-to-end one-way latency is higher than the link latency by 50 milliseconds to account for an Internet path. The link buffer is set to 7 packets for GPRS and WLAN, and to 20 packets for UMTS. The TCP agent is uni-directional TCP SACK with delayed acknowledgments, Limited Transmit, timestamps, and the receiver window of 50 segments (it is sufficiently large so that the protocol behavior is dominated by the bottleneck buffer in the network).

By default, TFRC history discounting is enabled, the feedback frequency is once per RTT, and self-clocking is disabled. Our simulation scripts are publicly available [16].

IV.B. TCP Simulation Results

In this section, we evaluate the performance of a single TCP connection during an ideal handover between GPRS and UMTS. After a handover from GPRS to UMTS in Figure 5(a), it takes approximately 10 seconds for the connection to fully utilize the new link. This delay is explained by the slow increase of the TCP window in congestion avoidance and the increased bandwidth-delay product of the path.

In Figure 5(b), a TCP flow creates congestion after a handover from UMTS to GPRS. The handover causes packet losses due to the reduced bandwidth-delay product of the path. TCP experiences a retransmit timeout due to many lost segments. For several seconds the TCP flow remains idle waiting for the retransmit timer to expire. In previous work [18], we proposed a TCP variant NewReno-SACK that better avoids retransmit timeouts than the standard Reno-SACK used in this simulation. It can improve TCP throughput in the presence of handovers, but cannot entirely eliminate delays due to the retransmissions of lost packets.

In summary, the self-clocking property of TCP allows a relatively rapid adaptation to changed link bandwidth after a handover. However, a retransmit timeout due to packet losses forces TCP to lose self-clocking. Furthermore, it is important to avoid a spurious retransmit timeout when RTT suddenly increases after a handover [18].

IV.C. TFRC Simulation Results

In this section, we evaluate the behavior of a single TFRC flow during an ideal handover between GPRS and UMTS. In Figure 6(a), a TFRC flow significantly underutilizes the UMTS link after a handover from GPRS. This behavior differs from measurements shown in Figure 3(a), where TFRC quickly increased the transmission rate. This discrepancy is explained by the presence of losses due to a buffer overflow at time 10 in Figure 6(a). They terminate TFRC slow start and prevent rapid acceleration after a handover. Furthermore, high latency of UMTS contributes to the slow increase of the transmission rate. Such sluggishness can prevent TFRC from using the available bandwidth if the fast link is available only for a short period of time.

In Figure 6(b), a TFRC flow creates heavy congestion after a handover from UMTS to GPRS. The TFRC flow starts to react to the reduced bandwidth only after several acknowledgments. Taking the interval at which acknowledgments are sent, about 50 data packets are transmitted and dropped before the sender starts slowing down. The flow slows down sufficiently only after 20 seconds from the handover. Such delays in the reduction of transmission rate can negatively affect concurrent traffic from other users and applications.

IV.D. TFRC Simulation Results with a Competing TCP Flow

In this section, we experiment with a TFRC flow with a competing TCP flow during an ideal handover between GPRS and UMTS. In Figure 7(a), the TFRC flow is shown at the upper and the TCP flow at the lower part of the graph. For convenience, sequence numbers wrap every 90 segments. In this scenario, the TCP flow receives 12 times more bandwidth than the TFRC flow. Such gross unfairness starts at time 5 after a burst of losses resulting from buffer overflow. While the TCP connection is able to recover from losses and transmit at a steady rate, the TFRC flow only transmits a packet every five seconds. After the handover at time 30, TCP adapts to the new link rate after ten seconds. The TFRC flow increases the rate very slowly and even at time 60 has not yet converged to the fair share of the link bandwidth.

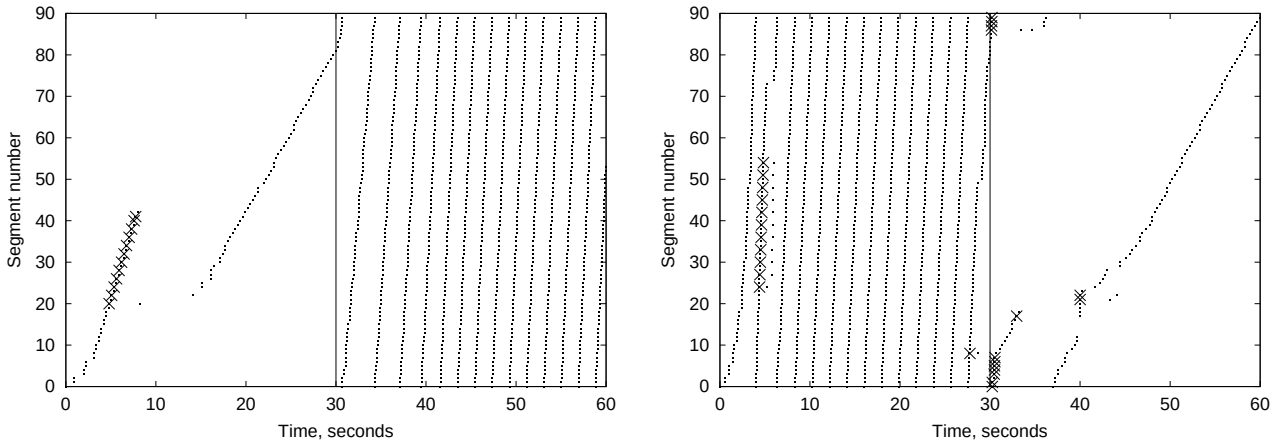
In Figure 7(b), TFRC and TCP flows are shown during a handover from UMTS to GPRS. TFRC reduces the rate excessively after a buffer overflow in UMTS at time 5, that results in TCP obtaining more bandwidth than the TFRC flow before the handover. However, after the handover, despite the high packet loss rate, TFRC transmits faster than TCP. In fact, a similar observation was made during measurements in Figure 4(b), where TFRC prevented a TCP flow from getting any packets through after a handover.

V. Effect of TFRC Parameters

In this section, we examine the effect of self-clocking, history discounting, and feedback frequency on aggressiveness and responsiveness of TFRC during a simulated ideal handover.

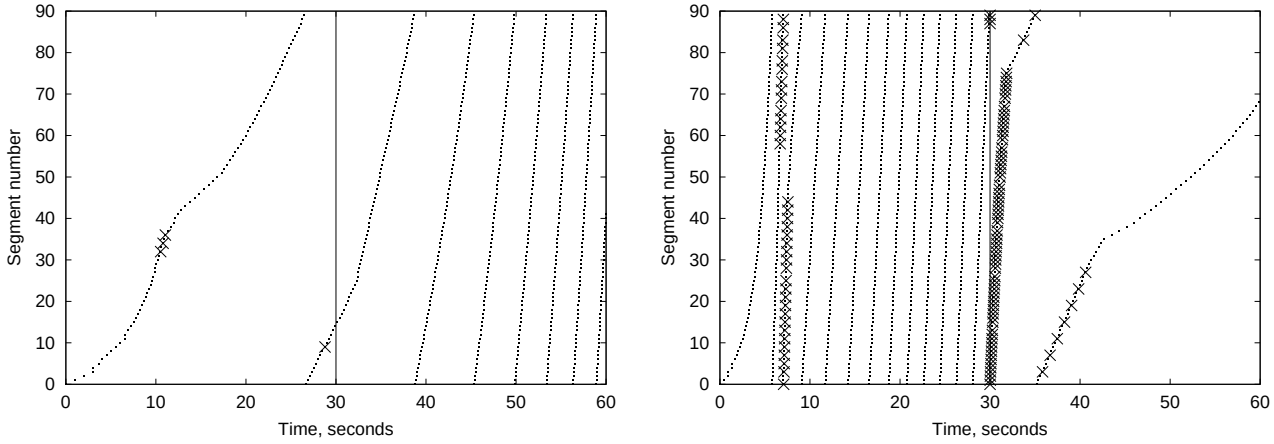
Self-clocking can improve TFRC responsiveness after a UMTS to GPRS vertical handover when the available bandwidth sharply decreases. Figure 6(b) showed TFRC behavior without self-clocking. With self-clocking, a TFRC flow reduces the rate somewhat faster after a handover; the transmission rate better corresponds to the actual link bandwidth. There are fewer congestion losses with self-clocking.

When history discounting is enabled, TFRC is able to forget about losses in the past faster. This is a useful feature for handovers from GPRS to UMTS when available bandwidth increases, especially when error losses occur during a handover. Figure 6(a) showed TFRC behavior with history discounting enabled. When we repeated the experiment without history discounting, TFRC was slightly more aggressive than with history discounting.



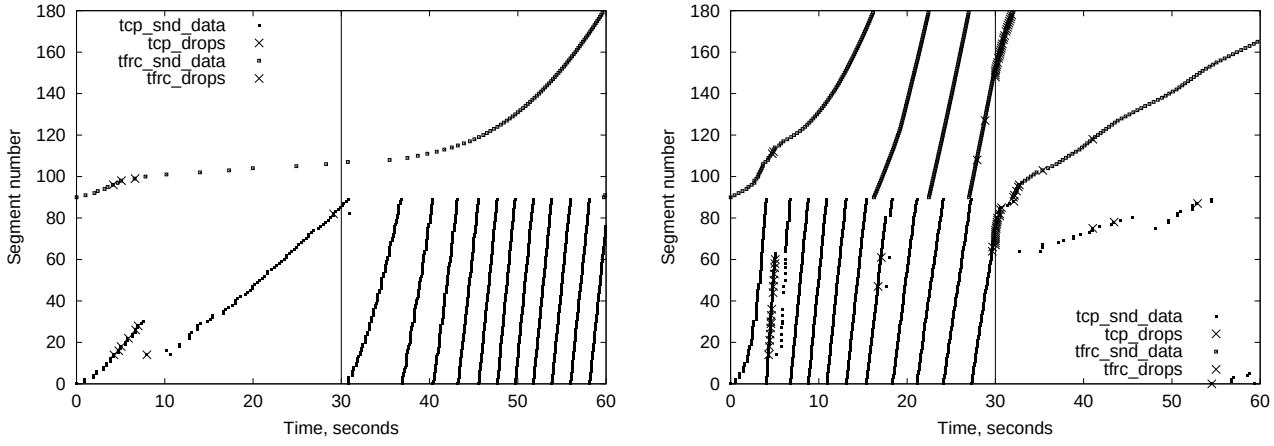
(a) GPRS → UMTS (b) UMTS → GPRS

Figure 5: Simulated behavior of a TCP flow during an ideal handover. A vertical line indicates the handover time. X marks show drops at the bottleneck queue.



(a) GPRS → UMTS (b) UMTS → GPRS

Figure 6: Simulated behavior of a TFRC flow during an ideal handover.



(a) GPRS → UMTS (b) UMTS → GPRS

Figure 7: Simulated behavior of a TFRC flow (top) with a competing TCP flow (bottom) during a vertical handover.

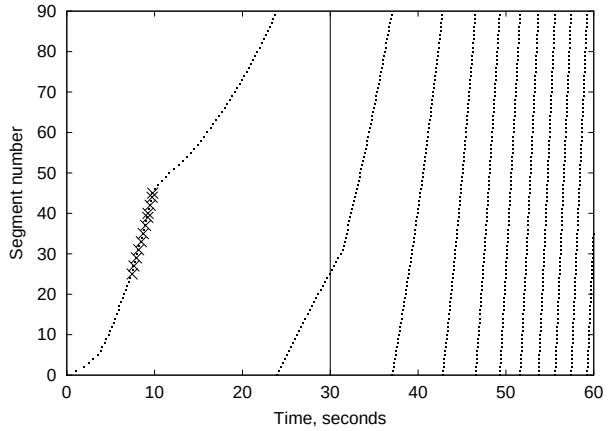


Figure 8: Effect of a higher feedback frequency (three acknowledgments per RTT) on TFRC during a handover from GPRS to UMTS.

Normally, TFRC uses feedback frequency of one packet per RTT. Higher feedback frequency makes it more resilient to loss of feedback packets, but should not significantly affect the dynamics of the protocol. Figure 8 shows TFRC with feedback frequency increased from one to three times per RTT. The result can be compared to Figure 6(a), where feedback frequency was once per RTT. Surprisingly, a shorter feedback interval improves aggressiveness and responsiveness TFRC by 30%. However, a scenario with a higher feedback frequency was shown where TFRC transmits at a lower rate, because several packets are lost in the beginning of a flow [17].

In summary, existing TFRC optimization mechanisms are helpful but not sufficient to adapt to changing link characteristics after a vertical handover. Even with an optimal choice of TFRC parameters, heavy congestion is present for several seconds after switching from a fast to a slow link. It can take tens of seconds for a TFRC flow to fully utilize a fast link after a handover from a slower link.

VI. Dealing with a Changing Bandwidth-Delay Product

The size of the link buffer is commonly set to the product of delay and bandwidth of the link. An interesting problem arises when a handover occurs between two networks with different bandwidth-delay products (e.g., GPRS and UMTS). When forwarding packets from a network with a high bandwidth-delay product to a low one, some data can be lost because the buffer space is insufficient to hold all packets. When transferring from a low bandwidth-delay product network to a high one, the number of buffered packets may not be enough to utilize the new link.

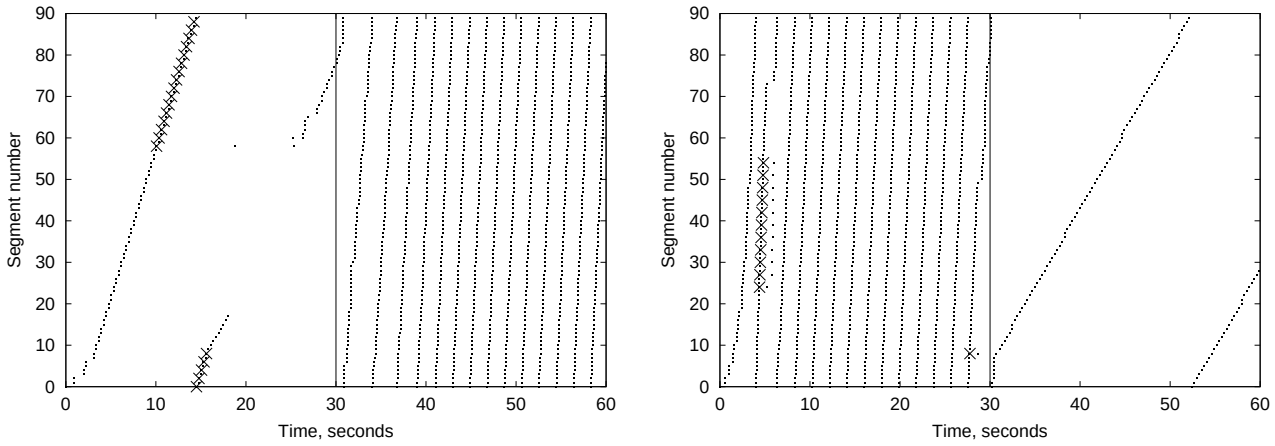
A possible solution to this problem can be configuring the buffer of all links to the maximum bandwidth-delay product of any link. Some links would become overbuffered, that is persistently have a longer queue than required for utilizing the link. However, packet losses or underutilization present after handovers can be reduced. A drawback of the proposed approach is a requirement to the network operator to know type of links that the user can handover to, which may not be always feasible in practice.

The effect of overbuffering on TCP flows can be seen in Figure 9(a) during a handover from GPRS to UMTS. In this simulation, the buffer size is fixed at 20 packets. TCP behavior can be compared to Figure 5(a) where the buffer size changes after the handover and the link is underutilized for four seconds. With overbuffering, a UMTS link is better utilized after a handover from the GPRS link. Figure 9(b) shows the effect of overbuffering after a handover from UMTS to GPRS. Packet losses that triggered a retransmission timeout in Figure 5(b) are eliminated and TCP performs optimally.

Overbuffering is known to have three negative aspects. First, interactive applications can suffer from the increased response time because of the queuing delay. In GPRS, the RTT is approximately 10 seconds with a buffer size of 10 kilobytes and is increasing by approximately one second per additional kilobyte. Second, the inflated RTT causes the retransmit timeout value at the sender to be very high delaying loss recovery. Third, when a data transfer is aborted, packets buffered in the network are unnecessarily delivered to the receiver.

It it planned that traffic in cellular systems is separated into different service classes [1]. Streaming and background traffic can use overbuffering without harming the interactive traffic, that solves the first problem. The second problem can be partly solved by implementing the state-of-the-art TCP at the end hosts, which is less prone to timeouts than the older TCP Reno. We proposed a solution to the third problem called Fast Reset that eliminates unnecessary data delivery from aborted data connections [15].

A scenario with overbuffering was shown where TFRC transmits at a higher rate after a handover, because the sender stays in slow start due to the absence of losses in the beginning of a flow [17]. We also found scenarios where TFRC does not benefit from overbuffering. The TFRC transmission rate is inversely proportional to the RTT. Reducing losses with overbuffering is compensated with increasing RTT



(a) GPRS → UMTS (b) UMTS → GPRS

Figure 9: Effect of overbuffering on TCP.

due to queuing. Therefore, overbuffering is a more useful mechanism for TCP than TFRC. The next section presents a highly efficient mechanism for TFRC.

VII. Explicit Handover Notification

In a highly dynamic network environment it is challenging for end-to-end protocols to estimate network characteristics accurately. Feedback from link layers that have local knowledge of the link conditions can be helpful to transport protocols [44]. Such mechanisms are currently under discussion in IETF (Trigtran) [24]. In this section, we examine how TCP and TFRC could utilize such information if it is made available to them.

To improve TCP performance for vertical handovers it can be helpful to artificially change the transmission rate of the sender. The TCP receiver is able to limit the transmission rate by manipulating the advertised window [37]. Additionally, by setting the Explicit Congestion Notification bit, the receiver can signal to the sender the need to reduce the transmission rate. As a last resort, the receiver can deliberately drop a packet to avoid heavy losses in the future. Using the receiver window also allows accelerating the sender. The TCP sender can grow the congestion window while being limited by the receiver window. By increasing the receiver window the TCP sender can be made to transmit at a higher rate.

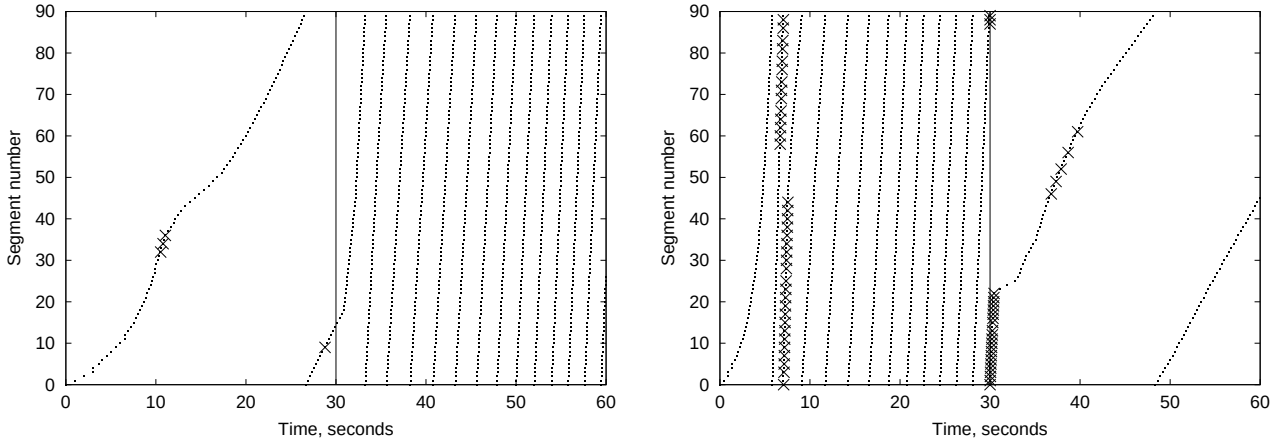
For slowly responsive congestion control, such as TFRC, the problem of adapting to varying network conditions is even more topical than for TCP. TFRC is forced to reduce the rate quickly during high loss rates to avoid heavy congestion. However, it is fairly slow to probe for available network bandwidth.

The TFRC receiver reports the estimated throughput and recent loss history to the sender. It is possible to adjust receiver reports to reflect changes in the networking conditions after a handover. TFRC implementations differ in how rapidly they increase the transmission rate when the calculated rate suddenly increases. The TFRC specification [20] makes possible an instant increase of the transmission rate to the rate given by the rate equation (however, the specification discourages increasing the rate more than twice per RTT to be compatible with TCP).

When receiving a handover notification from lower layers, the TFRC receiver could change the loss rate (p) and throughput estimates (X_{rcv}) in its standard reports according to characteristics of a new link for several RTTs (three in our tests). Consequently, the receiver reports real throughput and loss rate. These "faked reports" allow to instantly change the transmission rate of the sender and hide non-congestion related losses during a handover. Figure 10(a) shows the effect of the explicit handover notification on a TFRC flow after a handover from GPRS to UMTS. Underutilization on the UMTS link present in Figure 6(a) is eliminated. A TFRC flow with a handover notification in Figure 10(b) (from UMTS to GPRS) causes fewer losses than without it in Figure 6(b).

However, simply changing the receiver reports without adjusting the receiver state allows the transmission rate to restore when reports are not changed anymore. Indeed, TFRC keeps estimates of the loss rate, RTT, and throughput as smoothed averages. Plenty of new samples may be needed to change the average value so that it reflects new network characteristics. We found that resetting the TFRC receiver state after a handover eliminates this problem.

The explicit handover notification can be used when servers connect to the wireless network via a LAN



(a) GPRS → UMTS (b) UMTS → GPRS

Figure 10: Effect of explicit handover notification on TFRC.

with abundant transmission capacity. It is reasonable to expect that network operators place their real-time application servers as close to the user as possible to avoid extra latency of an Internet path.

It is an open question if the explicit handover notification for TFRC can be used in the public Internet. An abrupt increase in the transmission rate can cause transient congestion when the bottleneck link is somewhere else than in the wireless link. However, slow start in TCP causes a similar problem of transient congestion, but is widely accepted as a safe mechanism for the Internet.

VIII. Conclusions

We believe that vertical handovers are a fundamental property of future mobile networking. In this paper, we explored the effect of a change of networking characteristics triggered by vertical handovers on end-to-end transport protocols and arrived at the following results:

- Using measurements of handovers between GPRS and WLAN, as well as simulation of ideal handovers between GPRS and UMTS, we have shown that TFRC has significant difficulties in adapting to new link characteristics after a handover. In particular, TFRC receives only 10-50% of TCP thoughput over a fast link, while it can completely starve a TCP flow after handover to a slow link after a handover. The adaptation time of the TFRC rate to new link characteristics can be from tens to hundreds of seconds.

- Tuning TFRC parameters has only a minor positive effect. In particular, enabling self-clocking and history discounting in TFRC has slightly improved its responsiveness and aggressiveness. A higher feedback frequency from the TFRC receiver allows to increase the rate faster.

- We proposed and evaluated two mechanisms to improve transport performance during vertical handovers. With overbuffering, the bottleneck buffer of all links is set according to the maximum delay-bandwidth of any link. It helps TCP to smoothly change between links with different bandwidth-delay products. With an explicit handover notification, a TFRC receiver or a performance enhancing proxy adjusts TFRC feedback reports for several RTTs. It enables TFRC to quickly adapt to new link characteristics, while otherwise maintaining a smooth sending rate.

During experiments, we also made two important observations. First, implementing congestion control at the application layer may not be feasible, because UDP applications do not receive prompt congestion notification from the operating system. When we first run measurements with a standard Linux kernel, the user-level TFRC implementation had been grossly unfair to concurrent TCP flows. The problem was found in a local congestion notification inside the kernel. While TCP flows reduce the transmission rate upon filling of network buffers, TFRC flows continue to run at a high rate until detecting a packet loss. The second observation is that it is possible to define a set of TCP options that provides good performance in all overlay networks we considered. Thus, no modifications to TCP specifications to enable option renegotiation are needed.

The effort that designers of transport protocols are willing to spend for achieving good performance in

the presence of handovers needs motivation. If handovers occur only rarely, their negative effect on transport protocols could be ignored. However, there are scenarios, such as Infostations on a highway [13], where vertical handovers can be frequent. Therefore, we believe that the effect of vertical handovers on transport protocols is an issue of growing importance.

Acknowledgments

Antti Erkkilä, Olli Aalto, and Tomi Luostarinen helped with setting up the testbed and performing measurements. We also thank Mun Choon Chan for an idea on preventing packet reordering in ns-2, Pasi Sarolahti for consulting on Linux TCP, Mark Handley and Sally Floyd for valuable comments on details of TFRC.

References

[1] 3GPP. TS 23.107: QoS concept and architecture, Mar. 2002.

[2] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proc. of ACM SIGCOMM'99*, Aug. 1999.

[3] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. IETF RFC 2581, Apr. 1999.

[4] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM/Baltzer Wireless Networks*, 1(4):469–481, 1995.

[5] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *Proc. of ACM SIGCOMM'01*, Aug. 2001.

[6] D. Beaufort, L. Fay, C. Samson, and A. Teil. Measured performance of TCP friendly rate control protocol over a 2.5G network. In *Proc. of the IEEE Vehicular Technology Conference (VTC'02 Fall)*, Sept. 2002.

[7] C. Blondia, N. Van den Wijngaert, G. Willems, and O. Casals. Performance analysis of optimized smooth handoff in Mobile IP. In *Proc. of ACM MSWiM'02*, Sept. 2002.

[8] R. Cáceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, 1995.

[9] R. Caceres and V. N. Padmanabhan. Fast and scalable wireless handoffs in support of mobile internet audio. *ACM Mobile Networks and Applications*, 3(4):351–363, 1998.

[10] A. Fladenmuller and R. Silva. The effect of mobile IP handoffs on the performance of TCP. *ACM Mobile Networks and Applications*, 4(2):131–135, May 1999.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM'00*, Aug. 2000.

[12] D. Forsberg, J. Malinen, J. Malinen, T. Weckstrm, and M. Tiusanen. Distributing mobility agents hierarchically under frequent location updates. In *Proc. Sixth IEEE International Workshop on Mobile Multimedia Communications (MOMUC'99)*, Nov. 1999.

[13] R. H. Frenkiel, B. R. Badrinath, J. Borras, and R. D. Yates. The Infostations challenge: Balancing cost and ubiquity in delivering wireless data. *IEEE Personal Communications Magazine*, 7(2):66–71, Apr. 2000.

[14] A. Gurtov. Making TCP robust against delay spikes. Technical Report C-2001-53, University of Helsinki, Nov. 2001.

[15] A. Gurtov. Eliminating aborted data delivery over cellular links. *ACM Mobile Computing & Communications Review*, 7(4):53–54, Oct. 2003. Extended abstract (selected posters from Mobicom'03).

[16] A. Gurtov. Extensions of ns-2 simulator. Available at http://www.cs.helsinki.fi/u/gurtov/ns/, Mar. 2004.

[17] A. Gurtov and S. Floyd. Modeling wireless links for transport protocols. *ACM Computer Communication Review*, 34(2):85–96, Apr. 2004.

[18] A. Gurtov and R. Ludwig. Responding to spurious timeouts in TCP. In *Proc. of IEEE INFOCOM'03*, Apr. 2003.

[19] A. Gurtov, M. Passoja, O. Aalto, and M. Raitola. Multi-layer protocol tracing in a GPRS network. In *Proc. of the IEEE Vehicular Technology Conference (VTC'02 Fall)*, Sept. 2002.

[20] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. IETF RFC 3448, Jan. 2003.

[21] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proc. of ACM MOBICOM'03*, Sept. 2003.

[22] R. Hsieh and A. Seneviratne. A comparison of mechanisms for improving Mobile IP handoff latency for end-to-end TCP. In *Proc. of ACM MOBICOM'03*, Sept. 2003.

[23] ICIR. Equation-based congestion control for unicast applications, Aug. 2003. http://www.icir.org/tfrc/.

[24] IETF. Access link intermediaries assisting services BOF, Oct. 2003.

[25] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov. TCP over second (2.5G) and third (3G) generation wireless networks. IETF RFC 3481 (BCP 71), Feb. 2003.

[26] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM'88*, Aug. 1988.

[27] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: Congestion control without reliability. Available at http://www.icir.org/kohler/dccp/, May 2003.

[28] H. Levkowetz and S. Vaarala. Mobile IP traversal of network address translation (NAT) devices. IETF RFC 3519, May 2003.

[29] 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.

[30] P. Manzoni, D. Ghosal, and G. Serazzi. Impact of mobility on TCP/IP: an integrated performance study. *IEEE Journal on Selected Areas in Communications*, 13(5):858–867, 1995.

[31] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. IETF RFC 2018, Oct. 1996.

[32] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proc. of ACM SIGCOMM'98*, Sept. 1998.

[33] X. Perez-Costa, M. Torrent-Moreno, and H. Hartenstein. A performance comparison of Mobile IPv6, Hierarchical Mobile IPv6, fast handovers for Mobile IPv6 and their combination. *ACM Mobile Computing & Communications Review*, 7(4):5–19, Oct. 2003.

[34] C. Perkins. IP mobility support for IPv4. IETF RFC 3344, Aug. 2002.

[35] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. IETF RFC 3168, Sept. 2001.

[36] P. Sarolahti and A. Kuznetsov. Congestion control in linux TCP. In *Proc. of USENIX'02*, June 2002.

[37] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad. Receiver based management of low bandwidth access links. In *Proc. of IEEE INFOCOM'00*, Mar. 2000.

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

[39] W. R. Stevens. *TCP/IP Illustrated, Volume 1 (The Protocols)*. Addison-Wesley, Nov. 1994.

[40] UCB/LBNL/VINT. The ns-2 network simulator, Aug. 2003. http://www.isi.edu/nsnam/ns/.

[41] B. Walke. *Mobile Radio Networks, Networking and Protocols (2. Ed.)*. Wiley & Sons, 2001.

[42] H. J. Wang, R. H. Katz, and J. Giese. Policy-enabled handoffs across heterogeneous wireless networks. In *Proc. of the Second IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[43] J. Widmer, R. Denda, and M. Mauve. A survey on TCP-friendly congestion control. *IEEE Network*, 15(3):28–37, May 2001.

[44] G. Xylomenos. *Multi Service Link Layers: An Approach to Enhancing Internet Performance over Wireless Links*. PhD thesis, University of California at San Diego, 1999.

[45] Y. R. Yang, M. S. Kim, and S. S. Lam. Transient behaviors of TCP-friendly congestion control protocols. In *Proc. of IEEE INFOCOM'01*, Apr. 2001.

A. Static Protocol Options

Most connection-oriented transport protocols, such as TCP or DCCP, perform negotiation of protocol options during the connection establishment. In case of TCP, the options cannot be adjusted later during the connection lifetime. Options negotiated at the connection establishment may not be appropriate after a handover to the network with vastly different characteristics. In this section, we seek values of TCP options that would be appropriate for all considered overlay networks. Table 3 lists four widely implemented TCP options and a TCP header flag [35].

Table 3: Currently deployed TCP options.

Option Name	Values	Recommended Value
Timestamps	On/Off	On
Window scaling	Scale factor	4
MSS	Bytes	1460
SACK-enabled	On/Off	On
ECT (flag)	On/Off	On

The timestamp option requires that both connection end points use it through the connection lifetime. A timestamp and an echo of the received timestamp are placed in every segment. The benefit of always using the timestamp option is questioned [2] because of its 12-byte overhead in every segment. However, some studies found that timestamps are useful in the wireless environment [14]. Therefore, we believe that the use of the timestamp option is justified in all considered overlay networks.

The window scale option defines a multiplier for the receiver window. A scaled window is appropriate in networks with a high bandwidth-delay product, such as UMTS (with bandwidth close to 2 Mbps). However, limiting the receiver window to a smaller size is often beneficial in slow networks, such as GPRS, to prevent excessive queueing in the network [19]. Using the scaling option, the receiver window becomes of a granularity of 2, 4, 8, ... bytes for the scaling parameter of 1, 2, 3, ... Reduced granularity does not significantly affect the ability of the receiver to limit the size of the receiver window, if necessary. Hence, the receiver can negotiate the largest required scale factor even if the connection is initiated in the network with a low delay-bandwidth product, such as GPRS.

Justifications for setting the maximum segment size (MSS) were given, for example, by Stevens [39]. In summary, the trade-off is between lighter header overhead (with larger segments) and inefficient operation in the presence of packet losses and high latency. GPRS does not have high packet loss rates because of retransmissions at the link layer. In GPRS, the latency is already so high that using a large segment size does not significantly increase it. Therefore, using the MSS of 1460 bytes in all overlay networks is acceptable. A slightly smaller value should be used to avoid fragmentation due to tunneling by Mobile IP.

The SACK-enabled option [31] informs that the end point supports selective acknowledgments. The ECN-capable transport (ECT) flag defines that the end point understands an explicit congestion notification [35] given by routers. These options are useful in any network.

Hence, we found a set of option values adequate for all considered overlay networks. In fact, these values are approved as a best current practice recommendation in the IETF [25]. It is fortunate that the TCP protocol need not be modified to enable re-negotiation of options during an ongoing connection.

B. Preventing Packet Reordering

We implemented an algorithm in ns-2 to prevent packet reordering that can occur during a step change in link bandwidth and latency. The algorithm can be implemented in real-world networking nodes scheduling packets over multiple links or over a link with frequently changing bandwidth. An intuitive purpose of the algorithm is to avoid transmitting a packet if it could arrive to the receiver earlier than the previously sent packet.

$arrTimePrev = 0$
$for\ next\ pkt\ to\ send$
  $arrTime = pktSize/b + d$
  $if\ arrTimePrev < arrTime$
    $arrTime = arrTimePrev + pktSize/b$
  $arrTimePrev = arrTime$
  $schedulePkt(arrTime)$

Here b is the link bandwidth, d is the link one-way latency, and $arrTime$ estimates when the packet would arrive to the receiver across the link. Note that b and d can change during the execution of the algorithm. In ns-2, an arrival of the packet to the receiver is scheduled directly with schedulePkt(). If the algorithm is implemented in a real router, then schedulePkt() refers to transmission of the packet to the link and should be called with $arrTimePrev - s/b - d$ as a parameter.