Measurement and Analysis of TCP-Friendly Rate Control for Vertical Handovers

Andrei Gurtov*
University of Helsinki

Jouni Korhonen
TeliaSonera Finland

(*) Corresponding author:
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 UNIVERSITY OF HELSINKI
tel: +358 40 5963729, fax: +358 9191 44441, gurtov@cs.helsinki.fi

Draft 29.6.2003

Abstract

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 TFRC designed to provide a smooth transmission rate for real-time applications and therefore less responsive to changes in network conditions than TCP. In the first part of the paper, we measure performance of TFRC and TCP flows during a vertical handover between GPRS, WLAN, and LAN in our testbed based on Mobile IP. As expected, TFRC is less aggressive than TCP over fast links. However, after a handover to a slow link TFRC can starve a competing TCP flow. In the second part of the paper, we use a model of an *ideal handover* between UMTS, GPRS and WLAN. An ideal handover does not cause error losses, delays, packet duplication or reordering. These negative effects can be partly or completely eliminated by new handover algorithms; an abrupt change in latency and link bandwidth after a vertical handover cannot be avoided. As expected, TFRC can overshoot or underutilize the new link significantly after a handover. Tuning TFRC parameters including self-clocking, history discounting and higher feedback frequency from the receiver has a positive but insufficient effect on TFRC responsiveness and aggressiveness. We evaluate two new mechanisms to improve transport performance for vertical handovers: *overbuffering* and an *explicit handover notification.* Overbuffering prevents packet losses during handovers between links with a different bandwidth-delay product and is more useful for TCP than to TFRC. The explicit handover notification allows to instantly adapt the TFRC rate on a new link, which otherwise can take up to a minute.

Keywords: Transport protocol, equation-based congestion control, wireless overlay networks, intersystem mobility, Mobile IP.

1. Introduction

Different wireless networks offer to a mobile user a tradeoff between connection bandwidth, coverage and cost. The user can utilize a most suitable wireless network at a given time and location. The user can switch between different wireless networks while keeping ongoing data communications. 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 [30].

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 amount of buffering can change at once, 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 serving well many applications that require 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 [40][54]. 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) [20] is perhaps the most popular protocol among proposed alternatives.

This paper presents traces of TCP and TFRC flows recorded in a testbed during vertical handovers. Our testbed connects GPRS, UMTS, WLAN and LAN overlay networks using Mobile IP [42]. From the traces we are able to estimate the duration of handovers and the number of lost packets. We also look at how quickly TCP and TFRC flows adapt their transmission rate when path characteristics change after a vertical handover and whether the bandwidth is shared fairly between TCP and TFRC.

To verify our testbed measurements and to study closely the effect of a step change in path characteristics we use a simulation model in ns2 [39]. A model of an ideal handover[1] has limited packet losses and no delays or packet reordering as if a smooth handover with packet forwarding were implemented [9]. Essentially, an ideal handover is represented by a

[1] We avoid an overused term 'seamless handover'.

step change in the bottleneck link bandwidth, latency and buffer size. This model allows us to concentrate on fundamental aspects of a vertical handover that cannot be eliminated by a better implementation of a handover. We look at aggressiveness, responsiveness and fairness of TCP and TFRC flows after an ideal handover. The simulation results are compared with measurements results from the testbed.

The main observation from measurements and simulations is that it takes a long time for TCP and especially TFRC to utilize the available bandwidth after a handover. We first try existing TFRC mechanisms to improve the performance. Because we found them insufficient, we attempt two new approaches. With overbuffering, the size of the bottleneck buffer for all overlay networks is set to a larger value to prevent data loss during a handover. With explicit handover notification, feedback reports of the TFRC receiver are adjusted according to the available bandwidth after a handover.

The rest of the paper is organized as follows. Section 2 gives the necessary background on end-to-end transport protocols and congestion control. Section 3 gives a set of TCP options suitable for use in any of overlay networks in this paper. In Section 4 we describe our Mobile IP testbed and present TCP and TFRC traces from vertical handovers. In Section 5, behavior of TCP and TFRC during an ideal handover is explored via simulation. In Section 6 we examine the effect of TFRC parameters. In Section 7 and 8 we introduce and evaluate overbuffering and explicit handover notification for improving aggressiveness and responsiveness of TFRC and TCP. Finally, Section 9 presents main conclusions from this work.

2. Background and Related Work

In this section, we review congestion control mechanisms in TCP and TFRC. The cornerstone of congestion control in today's Internet is the assumption that a packet loss almost always indicates congestion in the network [28]. Newly developed mechanism such as Explicit Congestion Notification [43] cannot change this basic assumption, as it would require updating *all* routers in the Internet.

2.1 End-to-end Congestion Control

2.1.1 Transmission Control Protocol

TCP is a reliable transport protocol still responsible for over 90% of all traffic in the Internet. TCP invokes slow start in the beginning of connection and after a retransmission timeout [1]. In slow start, the Multiplicative Increase Multiplicative Decrease (MIMD) roughly doubles the transmission rate every RTT in the absence of congestion. In steady state, known as congestion avoidance, TCP invokes Additive Increase Multiplicative Decrease (AIMD) congestion control. In this phase the rate is slowly increased by one packet per RTT in the absence of congestion and halved when a packet loss is detected. TCP has an important property of self-clocking also known as the packet conservation principle [28]. In the equilibrium condition each arriving ACK indicates that a segment has left the network and triggers a transmission of a new segment.

For loss recovery, TCP invokes go-back-N behavior after a retransmission timeout. Fast retransmit enables quick recovery on duplicate acknowledgments. We assume the reader is familiar with these basic TCP mechanisms [48]. Recently standardized advanced error recovery techniques include selective acknowledgement [38][8] and the limited transmit algorithm [3]. Numerous experimental solutions have been proposed to improve TCP performance over lossy wireless links [4][13][15]. Major approaches include end-to-end adjustments, a split connection, and a TCP-aware link layer [6][5].

A study on the effect of mobility on TCP found that packet losses during a handover significantly reduce throughput [37]. The most significant factor contributing to long TCP recovery from handovers was found to be in the exponential backoff of the TCP retransmit timer [11]. A proposed solution is to artificially generate three DUPACKs at the TCP receiver to trigger fast retransmit at the sender and avoid waiting for next expiration of the retransmit timer. An improved variant of this mechanism is proposed in [19].

Delaying, reordering, and duplication of packets during a vertical handover can reduce TCP performance [13][56]. Packets are delayed until a mobile host connects and registers itself in a new wireless network. Packet reordering can occur when the latency of the new link is significantly less than of the old link. Packets sent after a handover on the new link arrive faster than packets sent on the old link before the handover. Packet duplication can result from bicasting of packets on two or more simultaneously available links. The Eifel algorithm [36][24] can be used to detect delayed, reordered or duplicated packets and prevent spurious retransmissions and false congestion control.

The TCP transmission rate is defined by the rate of returning ACKs. The state of TCP congestion control is determined by the available buffer space at the bottleneck link. It makes the size of the buffer an important factor affecting TCP performance. The link buffer is typically set according to the bandwidth-delay product of the link. Therefore, window-based protocols such as TCP are more sensitive to the change of bandwidth-delay product of the link than only of the link bandwidth. For example, TCP congestion control estimators (congestion window, 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.

2.1.2 TCP-Friendly Rate Control

TFRC allows an application to transmit at a steady rate that is typically within a factor of two from the TCP rate in the similar conditions [20]. 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. The TFRC receiver is responsible for reporting the loss event rate `p` and average receive rate `X_rcv` to the sender. The sender computes the reference transmission rate `X_calc` based on `p`, `X_rcv` and measured RTT based on an equation of the TCP rate in similar conditions [40]. The actual transmission rate `X` is set as follows [25]:

```
if (p > 0)
   calculate X_calc using the TCP throughput equation
      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, `t_mbi` represents the maximum back-off time, 64 sec by default, in persistent absence of feedback. If `p` is zero, no packet loss has yet been seen by the flow. In this phase, the sender emulates slow start of TCP by doubling the transmission rate every RTT.

Calculating the loss event rate rather than simply taking the packet loss rate is an important part of TFRC. TCP does not typically reduce the congestion window more than once per a window of data. 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 the 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. It is triggered when the current loss interval is at least twice larger than the average.

Self-clocking is seen as the key feature of TCP congestion control that contributes to the stability of the Internet [7]. An optional self-clocking mechanism for TFRC works for the RTT following a packet loss by limiting the sender's rate to at most the received rate in the previous round trip. 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 fairness, aggressiveness, responsiveness, and smoothness. 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.

The maximum increase of TFRC rate is estimated to be 0.14 packet per RTT and 0.22 packets per RTT with history discounting [20]. Furthermore, it takes four to eight RTTs for TFRC to halve its sending rate in presence of persistent congestion.

We explained in Section 2.1.1 that window-based protocols such as TCP are sensible 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 rather transmits at the relatively static rate. Therefore, TFRC is more sensible 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) [31].

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 [55][20][7] in several ways. First, we are interested in cellular networks where little degree of statistical multiplexing is present[2]. This allows us to concentrate on behavior of a single flow. Second, we consider the effect of varying RTT. Third, we evaluate changes in link bandwidth and latency of up to two orders of magnitude. We are not aware of measurement studies of TFRC over wireless links other than [10].

2.2 Overlay Networks

The terms *wireless overlay networks* and a *vertical handover* were introduced during the Bay Area Research Wireless Access Network project [30][47]. BARWAN included WaveLAN, Infrared, Richochet wireless networks and later a

[2] In cellular systems such as GPRS the bottleneck buffer is usually dedicated to one user. Wireless users typically have only a small number of concurrent connections due to limited bandwidth.

wide-area cellular network [53]. Other researchers built a number of similar testbeds, mostly concentrating on handovers between WLAN and LAN [26].

The main problem with implementing an intersystem handover is that transport connections in the standard TCP/IP stack are bound to use the same IP address through their lifetime. Many applications, such as HTTP without persistent connections, are transactional and have short data transfers. For them maintaining a transport-layer connection after a handover is not that important. However, many applications, such as streaming and FTP, use long-lived flows transferring large amounts of data. For such applications it is important that the transport connection stays alive after a handover, and preferably adapts quickly to changes in networking environment.

Several mechanisms were proposed to solve this problem. Mobile IP [42] assumes that the mobile host uses its permanent IP address from the home network at all times. Packets destined to this IP address are tunneled from the router in the home network (called a home agent) to the router in the visiting network (called a foreign agent). Other approaches [17][49] to implementing intersystem mobility can be classified into application-based (e.g. using the Domain Name System [45]), multicast-based [47], and micro mobility protocols with context transfers [32]. A good mobility protocol minimizes the delay and packet loss during a vertical handover.

Several studies evaluated performance of a Mobile IP 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 high to be tolerated by real-time applications [19][33]. A study of an optimized smooth handover in Mobile IP observed that forwarding packets from the old access point to the new one can significantly reduce packet losses [9].

In this paper, we consider four different overlay networks: GPRS, UMTS, WLAN and LAN. The *General Packet Radio Service* (GPRS) is an extension to GSM that provides higher data rates and 'always on' capability. GPRS is based on a packet-switched technology that permits efficient sharing of radio resources among users. GPRS allows a user to utilize multiple GSM timeslots that increases the data rate available to the user up tens of kilobits per second. The radio technology of the *Universal Mobile Telecommunication System* (UMTS) is based on code division multiple access with higher data rates and higher spectrum utilization than in GPRS. Data rates available to a user are in the range of 64 kbps to 2 Mbps. Initially, UMTS is being deployed for high-speed access in densely populated areas. *Wireless LANs* (WLAN) represent a number of systems operating in the license-free Industrial Scientific and Medical band. WLAN can provide data rates up to tens of megabits per second but its coverage is limited to tens of meters. Taxonomy of mobile radio networks can be found in [52].

Table 1. Link characteristics of overlay networks.

System	RTT, ms	Bandwidth, Mbps	Bandwidth*RTT, Kbytes	Coverage
GPRS	600	0.03	~2	country
UMTS	300	0.384	~15	city
WLAN	20	2	~5	building
LAN	5	10	~6	desk

Link characteristics of our overlay networks are summarized in Table 1. GPRS provides a low rate, high latency connection available on the all territory of the country[3]. UMTS provides higher data rates but only within cities. WLAN gives a high-speed connection in 'hot spot' locations such as airports or hotels. LAN is the 'fixed' way of connectivity by plugging into the wired Ethernet. Handovers between different systems cause a varying degree of change in link characteristics. For example, GPRS-UMTS handovers trigger a significant change in the delay-bandwidth product and GPRS-WLAN handovers a significant change in bandwidth but the delay-bandwidth product remains nearly the same because of the lower latency in WLAN.

3. Static Protocol Options

Most connection-oriented transport protocols such as TCP or DCCP [31] include 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 try to find values of TCP options that would be appropriate for any overlay network considered in this paper. Table 1 lists four widely implemented TCP options and a TCP header flag [43].

The timestamp option requires that both connection end points use it through the connection lifetime. A current timestamp and an echo of the received timestamp are placed in each segment. The benefit of always using the timestamp option is questioned [2] because it costs 12 Bytes of overhead in each segment. However, several studies found that

[3] Previous work often cites satellite systems as providing the widest overlay. However, direct satellite access has not yet seen significant success.

timestamps are useful in the wireless environment [56][22][24]. Therefore, we believe that use of the timestamp option is justified in all our overlay networks.

Table 2. Currently deployed TCP options.

Option Name	Value Range	Recommended Value
Timestamps	On/Off	On
Window scaling	Scale factor	4
MSS	Bytes	1500
SACK-enabled	On/Off	On
ECT (flag)	On/Off	On

The window scale option defines a multiplier for the receiver window. A scaled window is appropriate in networks with 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 buffering in the network [12]. Using the scaling option, the receiver window becomes of 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) are discussed in [48]. In summary, the tradeoff is between lighter header overhead (with larger segments) and inefficient operation in presence of packet losses and high latency. GPRS does not have high packet loss rates thanks to retransmissions at the link layer. In GPRS latency is already so high that using a large segment size does not significantly increase it. Therefore, using the MSS of 1500 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 [38] tells that the end point supports selective acknowledgments. The ECN-capable transport (ECT) flag defines that the end point understands explicit congestion notification [43] given by routers. These options are useful in any network.

Hence, we are able to identify option values adequate for all overlay networks considered in this paper. In fact, these values are approved as a best current practice recommendation in IETF [27]. It is fortunate that the TCP protocol need not be modified to enable re-negotiation of options during an ongoing connection.

4. Testbed Measurements

4.1 Methodology

Figure 1 shows the network architecture that was used during measurements. Mobile nodes can connect to the testbed using 100Mbps Ethernet LAN, 11Mbps 802.11b WLAN, a live GPRS network or live UMTS network[4].

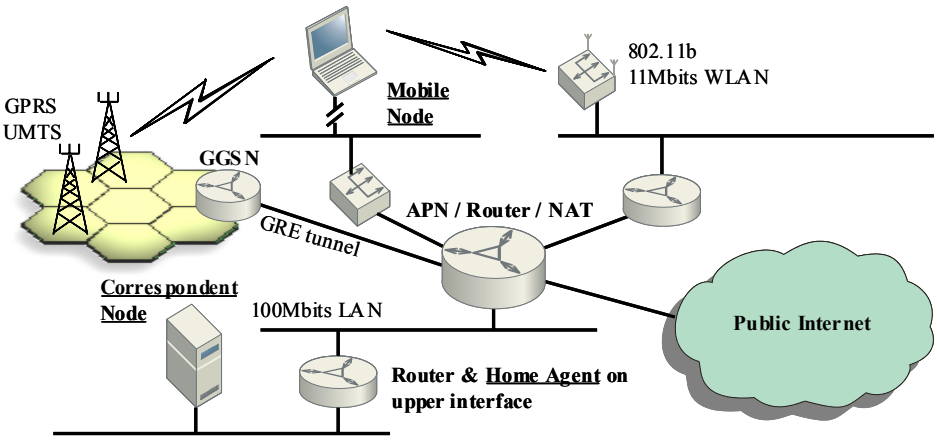


Figure 1. Testbed architecture based on Mobile IP.

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

[4] We do not present UMTS traces due to technical problems in the network at the time of submission of this paper.

The Mobile Node is a Toshiba Portégé 7200 with Pentium3 650MHz. The Correspondent Node and the Home Agent are Compaq Deskpros with Pentium3 600MHz. The APN router is an industrial PC equipped with Pentium3 600MHz. All machines are running the Linux operating system. The APN router has a Debian distribution with a 2.2.17 kernel. Both the Mobile Node and the Home Agent machines have Redhat 7.3 distributions with 2.4.17-3 kernels and finally the Correspondent Node has a RedHat 7.3 distribution with a 2.4.18-4 kernel.

For the GPRS access we used a Nokia's D211 PCMCIA card, which is capable of 3 downlink time slots and 1 uplink timeslot. 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 [16]. The SecGo Mobile IPv4 implementation is fully compliant to the latest RFC3344 [42] specification and also implements NAT Traversal (NATT) tunneling [34]. The Mobile IPv4 product we used does not implement any handover enhancements, such as a smooth handoff [9]. Buffering in the HA and in the MN does not modify standard Linux buffering.

In handover tests we used a co-located Foreign Agent mode in all networks. We forced reverse NATT tunneling and defined zero agent solicitations to be sent from the MN. These settings caused all traffic to go through the HA. During a handover the MN sends a registration request messages immediately to the HA using the newly available interface. Handovers were manually forced by changing the interface prioritization from the Mobile IP software's graphical interface. It may not be a practical scenario for vertical mobility, but is sufficient for our purposes of examining the behavior of end-to-end congestion control during vertical handovers.

NATT tunneling adds header overhead. Total header overhead is 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 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 from [18].

4.2 TCP Measurement Results

Table 3 summarizes delay and packet loss estimated from TCP traces during handovers in our testbed. No packet duplication or reordering was observed during the measurements. When all packets in downlink were lost, we did not have sufficient information to estimate losses in uplink.

Table 3. Delay and packet loss during handovers as measured in the testbed.

From→to	Total delay, sec	Loss in downlink, pkts	Loss in uplink, pkts
GPRS→LAN	13	all	-
LAN→GPRS	3	none	all
GPRS→WLAN	1	all	-
WLAN→GPRS	4	none	all
LAN→WLAN	0.2	none	all
WLAN→LAN	0.8	none	all

Figure 2 shows packet traces of TCP connections during vertical handovers. In Figure 2 (a) a GPRS to LAN handover causes a break in a TCP flow for 13 sec. The reason is an excessively high retransmission timeout value at the TCP sender in GPRS. The RTT in GPRS during a bulk data transfer is about 7 sec due to high link latency and significant amount of buffering [23]. During a vertical handover all outstanding packets get lost; it takes a long time before the TCP sender retransmits any data. Even if the handover itself causes a disruption much shorter, the TCP sender would not start using the new link quickly unless fast retransmit is triggered.

A LAN to GPRS handover in Figure 2 (b) lasted 3 sec. The TCP sender timed out and performed three retransmissions using exponential backoff. The retransmission timeout value is much lower in LAN than in GPRS due to much lower RTT. In fact, a minimum RTO value set at the sender can have a significant impact when the link handover time is lower. Linux uses the minimum RTO of 200 ms [44] while the recommended time in specifications is 1 sec [41]. A smaller minimum timeout value enables the TCP sender to discover the new available link faster. Interestingly, the first ACK that returns to the sender after handover at 24th sec acknowledges 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 ACKs except the one for the highest outstanding segment were lost. TCP generates an ACK for at least every second segment; if they arrived to the sender we would see unnecessary go-back-N retransmissions [36].

A GPRS to WLAN handover in Figure 2 (c) lasted 1 sec. It is much less than in Figure 2 (a) because a retransmission timeout is avoided. Although all data segments were lost, DUPACKs arriving after handover resume the connection quickly. Linux TCP uses the FACK algorithm [44] that enables triggering fast retransmit even after a single DUPACK rather than waiting for three DUPACKs required by the standard TCP[5] [1]. This feature improves the TCP sender's

[5] To prevent spurious retransmissions Linux disables the FACK algorithm when packet reordering is detected.

ability to avoid retransmission timeouts. In other repetitions of this test, we observed connection breaks of 17 sec when the TCP sender had to rely on the retransmission timeout to recover lost segments. The reason for the long delay was the same as in the GPRS to LAN case described above.

A WLAN to GPRS handover in Figure 2 (d) lasted 4 sec and the TCP sender made two retransmission attempts with exponential backoff. Again, as in the LAN to GPRS case, the first returning ACK acknowledges all packets outstanding before the handover. Handovers between LAN and WLAN in Figure 2 (e, f) took less than a second. In both cases, no data segments were lost in the downlink direction. The TCP sender experienced spurious retransmission timeouts in both cases.

Because we do not have access to the source code of the Mobile IP implementation used in experiments, we cannot explain reasons for delays or packet losses during handovers. For example, it would be interesting to find out why there are no packet losses in the downlink direction when a handover occurs from LAN or WLAN. According to specifications, the home agent did not perform any buffering during handovers.

4.3 TFRC Measurement Results

Figure 3 shows packet traces of TFRC flows during vertical handovers. TFRC aggressiveness can be evaluated from Figure 3 (a, c, f) when a handover is from a slower to a faster network. Aggressiveness during a handover from WLAN to LAN is good. However, when a handover is initiated from GPRS, past history of high RTT and low bandwidth slows down TFRC adaptation.

Figure 3 (b) shows an increasing amount of outstanding data in the GPRS network after a handover from LAN to GPRS. This is possible because of an excessively large buffer in current GPRS networks [23]. At the 40th sec in Figure 3 (b) there is more than 100 segments buffered in the network. TFRC can suffer from overbuffering more than TCP, because TCP has a receiver window that limits the amount of outstanding data in the network.

Responsiveness of TFRC to a decrease in bandwidth can be seen in Figure 3 (d, e). At 30th sec feedback packets stop arriving to the sender. About a second later, no feedback timer expires at the sender and the transmission rate is halved several times until the first feedback packet arrives at 40th sec. Because TFRC is a rate-based protocol, there is no visible break in transmission during a handover as with TCP in Figure 2 (d).

4.4 TFRC Measurement Results with a Competing TCP Flow

Figure 4 shows packet traces of a TFRC flow during vertical handovers with a competing TCP flow. In these experiments, a TCP flow is started first and followed by a TFRC flow after 3 sec. The handover is triggered at 30th sec.

In Figure 4 (a, c) a TFRC flow accelerates slower than a TCP flow after a handover. A TFRC flow gets about six times less bandwidth than a competing TCP flow. This result is expected because TFRC probes for available bandwidth slower than TCP.

In Figure 4 (b, d) after a handover the TCP flow is completely starved performing retransmissions using an exponential backoff. 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 we used for measurements has the minimum sending rate of one packet per RTT as suggested in [18]. On a slow GPRS link, a TFRC flow transmitting at this minimum rate can starve a TCP flow that employs an exponential backoff of up to 64 sec between retransmission attempts. The latest TFRC specification [25] requires a similar type of behavior as in TCP. Another possible explanation can be an excessive buffer in GPRS.

In Figure 4 (e, f) TFRC shares bandwidth fairly with TCP in WLAN. In LAN TCP receives three times more bandwidth than TFRC.

5. Simulated Ideal Handover

In this section we evaluate how TCP and TFRC flows react to an abrupt change in link bandwidth, latency, and buffer size after a vertical handover.

5.1 Methodology

In this section we want to focus on changing link characteristics and not on disruptions caused by a specific handover mechanism. Therefore, a simple approach presenting a handover as a step change in bottleneck link bandwidth, latency and the buffer size is sufficient for our purposes. We model an ideal handover using enhancements to the ns2 simulator [39]. We implemented an algorithm described in Appendix A to prevent packet reordering during a handover. The implementation of the Drop-Tail queue was enhanced to check for buffer overflow when the maximum queue size changes.

Our network topology is a simple dumbbell as described in [24]. Traffic is generated by single-directional downlink transfers. A handover is triggered on 100th sec after the flow is started, the simulation is stopped on the 200th sec. The bottleneck queue is Drop-Tail; in previous work we found that RED performs poorly when there is no statistical multiplexing on the link [21]. The bandwidth and one-way latency of the link are set according to Table 1. The one-way latency between the end points was higher than the link latency by 50 ms to account for an Internet path. The link buffer

is set to 7 packets for GPRS and WLAN, and to 25 packets for UMTS. The TCP agent is one-directional TCP SACK with delayed acknowledgements, Limited Transmit, timestamps and the receiver window of 50 segments. By default, TFRC history discounting is enabled, the feedback frequency is once per RTT, and self-clocking is disabled.

5.2 TCP Simulation Results

Figure 5 shows behavior of a single TCP connection during an ideal handover between GPRS, UMTS and WLAN. After a handover from GPRS to UMTS in Figure 5 (a) it takes approximately 10 sec for the connection to utilize the new link. For a handover from GPRS to WLAN in Figure 5 (c) this time is only 3 sec. Although the difference in bandwidth is higher between GPRS and WLAN than between GPRS and UMTS, lower latency of WLAN enables a TCP flow to increase the rate significantly faster than in UMTS.

In Figure 5 (e) a handover from UMTS to WLAN is shown. Because of the higher delay-bandwidth product in UMTS, many packets are lost when the link buffer is resized according to the lower delay-bandwidth product of WLAN. However, those packet drops do not slow down the TCP flow in WLAN. In Figure 5 (f) a TCP flow performs well during a WLAN to UMTS handover.

In Figure 5 (b, d) a TCP flow creates congestion after handovers from UMTS and WLAN to GPRS. For a handover from WLAN to GPRS in Figure 5 (d), only two packets get dropped because the bandwidth-delay product of both links is similar. A UMTS to GPRS handover causes many packet losses due to the changed bandwidth-delay product. In Figure 5 (b) TCP experiences a retransmission timeout due to many lost segments. For 10 sec the TCP flow remains idle waiting for the retransmission timer to expire. In previous work [24] we proposed a TCP variant NewReno-SACK that better avoids retransmission timeouts than the standard Reno-SACK used in this paper.

The self-clocking property of TCP allows quick adaptation to changed link bandwidth after a handover. A retransmission timeout forces TCP to loose self-clocking. It is important that the retransmit timer at the TCP sender is restarted on ACKs [2] to avoid a spurious retransmission timeout when RTT suddenly increases.

5.3 TFRC Simulation Results

Figure 6 shows behavior of a single TFRC flow during an ideal handover between GPRS, UMTS and WLAN. In Figure 6 (a, c) a handover from GPRS to UMTS and WLAN causes significant underutilization of the link. In case of a GPRS to UMTS handover, high latency of UMTS contributes to the slow increase of the transmission rate. In Figure 6 (e) after a UMTS to WLAN handover it takes approximately 13 sec for a TFRC flow to reach the full rate.

In Figure 6 (b, d) a handover from UMTS to GPRS and WLAN create heavy congestion. The TFRC flow starts to react to it after several ACKs. Taking the interval at which ACKs are sent in Figure 6 (b), about 50 data packets are transmitted and dropped before the sender starts slowing down. The flow slows down sufficiently only after 20 sec from the handover.

In Figure 6 (f) during a WLAN to UMTS handover a TFRC flow is less responsive than TCP in Figure 5 (f). Congestion is created by TFRC because transmission continues at a high rate for some time. A larger buffer in UMTS is not sufficient to accommodate the higher sending rate until the TFRC flow slows down.

5.4 TFRC Simulation Results with a Competing TCP Flow

Figure 7 shows simulated behavior of a single TFRC flow with a single competing TCP flow during an ideal handover between GPRS, UMTS and WLAN.

In general, a TFRC flow is more aggressive than a TCP flow in these tests. TFRC receives up to two times more throughput than a TCP flow, except over UMTS in Figure 7 (a, c). When we rerun those tests with disabling of delayed acknowledgments in TCP, the result was the opposite. A TCP flow received more bandwidth than a TFRC flow. Because in differences among existing TCPs, designers of TFRC chose to make it slightly more aggressive than a typical TCP flow. However, over links with a high bandwidth-delay product such as UMTS, TFRC looses to TCP because of the slow increase in transmission rate.

6. Effect of TFRC Parameters

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

Figure 8 shows responsiveness of TFRC with self-clocking during a handover from UMTS to GPRS. It can be compared with Figure 6 (b) where TFRC without self-clocking was used in the similar experiment. With self-clocking, a TFRC flow reduces the rate faster after a handover. The transmission rate of the TFRC flow better corresponds to the actual link bandwidth when self-clocking is enabled. There are fewer congestion losses with self-clocking.

History discounting is a mechanism that improves TFRC aggressiveness in the absence of congestion. In Figure 9 history discounting is disabled during a handover from GPRS to UMTS. Figure 6 (a) showed TFRC behavior with history discounting enabled. When history discounting is enabled TFRC is able to forget about losses in the past faster. This is a useful feature especially when error losses occur during a handover before switching to a faster network. In this experiment TFRC is only slightly more aggressive when history discounting is enabled because there are few losses.

Figure 10 shows TFRC with feedback frequency increased from one to three times per RTT. The result can be compared to Figure 6 (a, b). A shorter feedback interval improves aggressiveness and responsiveness of TFRC. In Figure 10 (a) a TFRC flow is able to send 1200 packets at 130 sec; a flow with default feedback frequency in Figure 6 (a) transmits less than 800 packets. In Figure 10 (b) and in Figure 6 (b) a TFRC flow starts reducing the rate after receiving several ACKs after a handover. However, in Figure 10 (b) the interval between ACKs is shorter. There are fewer congestion losses with higher feedback frequency.

In summary, the standard TFRC optimization mechanisms are helpful but not sufficient to adapt to changing conditions after a vertical handover. Several seconds of heavy congestion remain after switching from a fast to a slow link. It takes tens of seconds for a TFRC flow to utilize a faster link after a handover from a slower link.

7. Overbuffering: Dealing with 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 vastly different bandwidth-delay products. 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*, i.e. persistently have a longer queue than required for utilizing the link. The effect of overbuffering on TCP and TFRC flows can be seen in Figure 11. TCP behavior can be compared to Figure 5 (a) where the buffer size changes after the handover. In this test the buffer size is constant at 25 packets. Underutilization present in Figure 5 (a) is eliminated. The UMTS link with a higher bandwidth-delay product is immediately utilized after a handover from the overbuffered GPRS link.

Aggressiveness of a TFRC flow in Figure 11 (b) is slightly improved compared to Figure 6 (a). However, the TFRC transmission rate is inversely proportional to the RTT. Reducing losses with overbuffering is compensated with increasing RTT due to queuing. Therefore, overbuffering is a more useful mechanism for TCP than TFRC.

Overbuffering is known to have three negative aspects. First, interactive applications can suffer from the increased response time because of the queuing delay. Second, the inflated RTT causes the retransmission timeout value at the sender to be very high that can delay loss recovery. Third, when a data transfer is aborted, a lot of packets buffered in the network are unnecessary delivered to the receiver. Priority scheduling of interactive traffic can solve 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 novel solution to the third problem called *Fast Reset.* It eliminates unnecessary data delivery from aborted data connections. The detailed description of the algorithm is given in Appendix B.

8. 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. Such mechanisms are currently under discussions in the Trigran working group in IETF [14]. 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 [46]. 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. It is because TCP-friendly congestion control 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 [25] allows an instant increase of the transmission rate to the rate given by the rate equation. We apply this idea to speed up a TFRC flow during a handover from GPRS to UMTS.

When receiving a handover notification from lower layers, the TFRC receiver changes the loss rate and throughput estimate in reports according to characteristics of a new link for three RTTs. After that, 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 12 (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 12 (b) causes fewer losses than without it in Figure 6 (b) after a handover from UMTS to GPRS.

During experiments we noticed that 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. A lot 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 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 connection.

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.

9. Conclusions

The first part of the paper presents measurements of TCP and TFRC flows over GPRS, WLAN and LAN in a testbed based on Mobile IP. Tracing of TFRC and TCP flows during vertical handovers gave following results:

- It is possible to define a set of TCP options that provides optimal performance in all overlay networks we considered. Similar analysis should be carried out for other transport protocols as well.
- Vertical handovers between GPRS, WLAN and LAN are demonstrated using Mobile IP. Handovers last from 200 ms up to several seconds, which is suitable for reliable flows but can be a problem for real-time flows.
- Implementing congestion control at the application layer may not be feasible. When we first run the tests, TFRC was unfair to a competing TCP connection. The reason was found in the 'local ECN' property of the Linux kernel[6] [50]. We used a TFRC implementation as an application on top of UDP. It does not reduce the transmission rate upon filling of a network buffer while a TCP flow does. Even a congestion-sensitive UDP application can be unfair to concurrent TCP flows when it sees no packet losses.
- TFRC suffers from deep buffering in the network because it has no receiver window. In TCP the receiver window limits the amount of outstanding data. In GPRS, where the bottleneck buffer is large, a TFRC flow gets about 100 segments outstanding in the network when the link delay-bandwidth product corresponds to 2-3 segments.
- TCP in GPRS has high RTO value of over than 10 s, which excessively delays using a new link unless fast retransmit is triggered. A lower minimum RTO value and forward acknowledgements in TCP can reduce a break in transmission due to handovers.

In the second part of the paper, we experimented with TFRC and TCP using a model of an ideal handover in ns2. This model concentrates on the fundamental effect of a vertical handover on end-to-end congestion control: changing bandwidth and latency. Based on handover simulations between UMTS, GPRS and WLAN we can make following conclusions:

- As expected, TFRC is slow to react to changes in link characteristics after a handover, causing overshooting or underutilization of the link. For example, it takes one minute to reach the full bandwidth after a handover from GPRS to UMTS.
- TFRC interprets packet losses during a handover as a single loss event and is not very sensitive to them. Therefore, preventing data loss during a handover may not be crucial to performance.
- TCP experiences retransmission timeouts due to many lost packets after a handover to a slower link.
- Enabling self-clocking, history discounting and higher feedback frequency in TFRC has a positive but generally minor effect. A higher feedback frequency allows TFRC to increase the rate significantly faster.

We proposed and evaluated two mechanisms to improve TFRC and TCP performance during vertical handovers. With *overbuffering*, the bottleneck buffer of all links is set according to the maximum delay-bandwidth of any link. A buffer overflow after a handover is eliminated. For TCP flows this allows to utilize the bandwidth of a new link much faster. We also proposed a novel algorithm named Fast Reset to eliminate a negative effect of overbuffering due to unnecessary delivering of data from aborted connections.

With *explicit handover notification* a TFRC receiver or a performance enhancing proxy adjusts feedback reports for several RTTs. When a new link is faster than the previous link, reporting error losses is suppressed and the receive rate is set according to the new link's rate. When a new link is slower, then the average loss rate is artificially increased to force the sender to reduce the rate faster. In simulations the explicit handover notification adapts a TFRC flow quickly to new link characteristics. Further research is needed to determine whether this is an appropriate mechanism to be deployed in the Internet. However, it can be used when a server is located in a LAN close to wireless networks.

[6] We disabled local ECN for all tests presented in this paper.

Acknowledgments

Antti Erkkilä, Olli Aalto and Tomi Luostarinen helped with setting up the testbed and doing measurements. Pasi Sarolahti offered comments on TCP implementation in Linux. Thanks to Mun Choon Chan for an idea on preventing packet reordering in ns2 and to Mark Handley and Sally Floyd for valuable details of TFRC.

References

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

[2] M. Allman, V. Paxson, On Estimating End-to-End Network Path Properties, ACM SIGCOMM, September 1999.

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

[4] A. Bakre, B.R. Badrinath, Handoff and systems support for indirect TCP/IP, In Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing, April 1995.

[5] H. Balakrishnan, S. Seshan, R. Katz, Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks, ACM Wireless Networks, Vol. 1, No. 4, December 1995.

[6] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, A Comparison of Mechanisms for Improving TCP Performance over Wireless Links, IEEE-ACM Transactions on Networking, Vol 5, No 6, 1997.

[7] D. Bansal, H. Balakrishnan, S. Floyd, S. Shenker, Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms, SIGCOMM 2001.

[8] E. Blanton, M. Allman, K. Fall, L. Wang, A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP, RFC3517, April 2003.

[9] C. Blondia, N. Wijngaert, G. Willems, O. Casals, Performance Analysis of Optimized Smooth Handoff in Mobile IP, MSWiM'02, September 2002.

[10] D. Beaufort, L. Fay, C. Samson, A. Teil, Measured Performance of TCP Friendly Rate Control Protocol over a 2.5G Network, IEEE Vehicular Technology Conference, Fall 2002.

[11] R. Cáceres, L. Iftode, Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments, IEEE Journal on Selected Areas in Communications, Vol. 13, No. 5, June 1995.

[12] R. Chakravorty, S. Katti, J. Crowcroft, I. Pratt, Flow Aggregation for Enhanced TCP over Wide Area Wireless, IEEE INFOCOM 2003, April 2003.

[13] M. Chan, R. Ramjee, TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation, MOBICOM 2002.

[14] S. Dawkins, C. Williams, A. Yegin, Problem Statement for Triggers for Transport (TRIGTRAN), draft-dawkins-trigtran-probstmt-01.txt, June 2003. Work in progress.

[15] D. Dutta, Y. Zhang , An Active Proxy Based Architecture for TCP in Heterogeneous Variable Bandwidth Networks, IEEE GLOBECOM 2001.

[16] Dynamics project, http://www.cs.hut.fi/Research/Dynamics, January 2003.

[17] M. Endler, V. Nagamuta, General Approaches for Implementing Seamless Handover, In Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, October 2002.

[18] Equation-Based Congestion Control for Unicast Applications, http://www.icir.org/tfrc/, January 2003.

[19] A. Fladenmuller, R. Silva, The Effect of Mobile IP Handoffs on the Performance of TCP, MONET, Vol 4, No 2, 1999.

[20] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control for Unicast Applications, ACM SIGCOMM, August 2000.

[21] A. Gurtov, TCP Performance in the Presence of Congestion and Corruption Losses, Master's Thesis, University of Helsinki, Department of Computer Science, Helsinki, December 2000.

[22] A. Gurtov, Making TCP Robust Against Delay Spikes, University of Helsinki, Department of Computer Science, Series of Publications C, No C-2001-53, November 2001.

[23] A. Gurtov, M. Passoja, O. Aalto, M. Raitola, Multi-Layer Protocol Tracing in a GPRS Network, In Proceedings of IEEE Vehicular Technology Conference (VTC'02), September 2002.

[24] A. Gurtov, R. Ludwig, Responding to Spurious Timeouts in TCP, IEEE INFOCOM'03.

[25] M. Handley, J. Pahdye, S. Floyd, J. Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, RFC 3448, January 2003.

[26] S. Helal, C. Lee, Y. Zhang, G. Richard, An Architecture for Wireless LAN/WAN Integration, IEEE Wireless Communications and Networking Conference (WCNC), 2000.

[27] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, F. Khafizov, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks, RFC 3481 (BCP 71), February 2003.

[28] V. Jacobson, Congestion Avoidance and Control, ACM SIGCOMM, August 1988.

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

[30] R. Katz, E. Brewer, The Case for Wireless Overlay Networks, SPIE Multimedia and Networking Conference (MMNC'96), January 1996.

[31] E. Kohler, M. Handley, S. Floyd, J. Padhye, Datagram Congestion Control Protocol (DCCP), draft-ietf-dccp-spec-02.txt, work in progress.

[32] R. Koodli, C. Perkins, Fast Handovers and Context Transfers in Mobile Networks, ACM CCR, October 2001.

[33] M. Kraner, C. Yap, S. Cvetkovic, E. Sanchez, Can Mobile IP Be Used As A Link Between IMT2000 Technologies And Operators, Workshop on Software Radios, University of Karlsruhe, March 2000.

[34] H. Levkowetz, S. Vaarala, Mobile IP NAT/NAPT Traversal using UDP Tunnelling, draft-ietf-mobileip-nat-traversal-07.txt, work in progress, January 2003.

[35] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, A. Joseph, Multi-Layer Tracing of TCP over a Reliable Wireless Link, ACM SIGMETRICS, 1999.

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

[37] 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, June 1995.

[38] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgement Options, RFC 2018, October 1996.

[39] Network Simulator ns2, http://www.isi.edu/nsnam/ns, January 2003.

[40] J. Padhye, V. Firoiu, D. Towsley, J. Kurose. Modeling TCP throughput: a simple model and its empirical validation, ACM SIGCOMM 1998.

[41] V. Paxson, M. Allman, Computing TCP's Retransmission Timer, RFC 2988, November 2000.

[42] C. Perkins, IP Mobility Support for IPv4, RFC3344.

[43] K. Ramakrishnan, S. Floyd, D. Black, The Addition of Explicit Congestion Notification (ECN) to IP, RFC 3168, September 2001.

[44] P. Sarolahti, A. Kuznetsov, Congestion Control in Linux TCP, In Proceedings of USENIX'02, June 2002.

[45] A. Snoeren, H. Balakrishnan, An end-to-end approach to host mobility, MOBICOM 2000.

[46] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, B. Bershad, Receiver Based Management of Low Bandwidth Access Links, IEEE INFOCOM 2000.

[47] M. Stemm, R. Katz, Vertical handoffs in wireless overlay networks, ACM Mobile Networks and Applications, Vol 3, No 4, 1999.

[48] W. R. Stevens, TCP/IP Illustrated, Volume 1 (The Protocols), Addison Wesley, November 1994.

[49] L. Taylor, R. Titmuss, C. Lebre, The Challenges of Seamless Handover in Future Mobile Multimedia Networks, IEEE Personal Communications, April 1999.

[50] B. Tierney, Using NetLogger and Web100 for TCP Analysis, In Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks, February 2003.

[51] P. Venkataram, R. Rajavelsamy, S. Laxman, A Method of Data Transfer Control During Handoffs in Mobile-IP Based Multimedia Networks, ACM Mobile Computing and Communications Review, Vol. 5, No 2, 2001.

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

[53] H. Wang, R. Katz, J. Giese, Policy-Enabled Handoffs Across Heterogeneous Wireless Networks, In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, February 1999.

[54] J. Widmer, R. Denda, M. Mauve, A Survey on TCP-Friendly Congestion Control, Special Issue of the IEEE Network Magazine on Control of Best Effort Traffic, 2001.

[55] Y. Yang, M. Kim, S. Lam, Transient behaviors of TCP-friendly congestion control protocols, In Proceedings of IEEE INFOCOM 2001, April 2001.

[56] M. Yavuz, F. Khafizov, TCP over Wireless Links with Variable Bandwidth, In Proceedings of the IEEE Vehicular Technology Conference, September 2002.

Appendix

A. Preventing Packet Reordering

We implemented an algorithm in ns2 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 [56]. 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.

```
arr_time_prev = 0
for next pkt to send
    arr_time = pkt_size/b + d
    if arr_time_prev < arr_time
            arr_time = arr_time_prev + pkt_size/b
    arr_time_prev = arr_time
    schedule_pkt(arr_time)
```

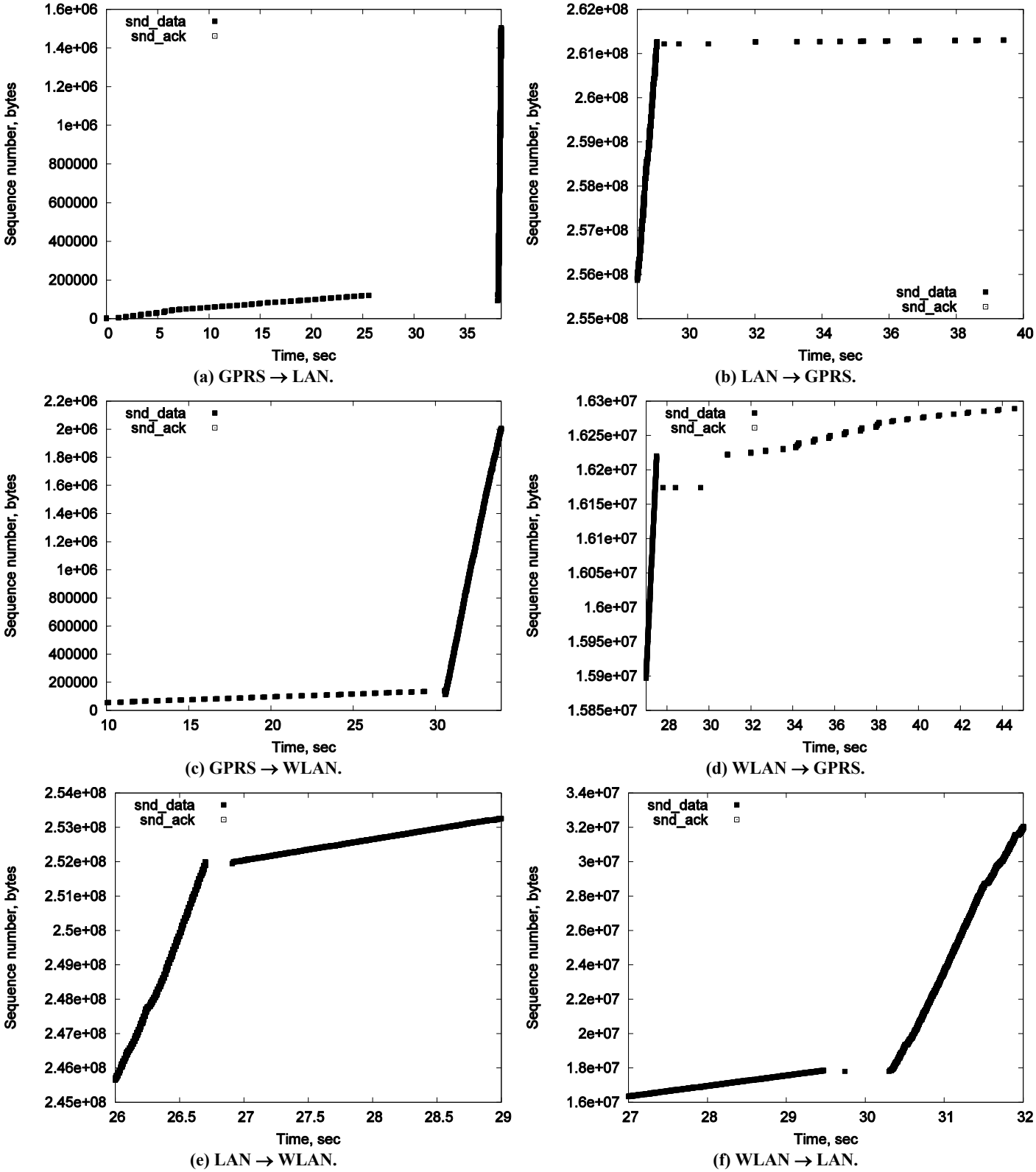
Here `b` is link bandwidth, `d` is link one-way latency, and `arr_time_prev` estimates when the packet would arrive to the receiver across the link. Note that `b` and `d` can change during execution of the algorithm. In ns2 an event of arrival of the packet to the receiver is scheduled directly with `schedule_pkt()`. If the algorithm is implemented in a real router, then `schedule_pkt()` refers to transmission of the packet to the link and should be called with `arr_time_prev - s/b - d` as a parameter.

B. Fast Reset Algorithm

This algorithm implements discarding of data from aborted data flows. It is suitable for any connection-oriented transport protocol that has reset packets (RST). The algorithm can be included into a performance enhancing proxy snooping packet headers. The limitation of the algorithm is in its usefulness only for symmetric routing, i.e. when packets from both directions flow through the same router.

```
for each arriving pkt
  if pkt is RST
    for each queued pkt2 in the other direction
      if (src, dst, sport, dport, prot) == (dst, src, dport, sport, prot)
            discard(pkt2)
forward(pkt)
```

The 5-tuple `(src, dst, sport, dport, prot)` uniquely identifies an IP flow by its source and destination IP addresses together with source and destination port numbers and the protocol number. A possible caveat is that the RST packet could be lost on the way to the sender after passing through the performance enhancing proxy. The sender in such case would be unaware that the receiver aborted the connection. For reliable protocols such as TCP, the algorithm relies on the retransmission timeout at the sender to resend the oldest outstanding segment. This segment will trigger another RST at the receiver. Hence, the sender is robustly notified of the connection reset due to the persistent retransmissions at TCP sender. Unreliable transport protocols typically include timers at the connection end points for discarding connection state in case of loss of reset packets.



(a) GPRS → LAN.

(b) LAN → GPRS.

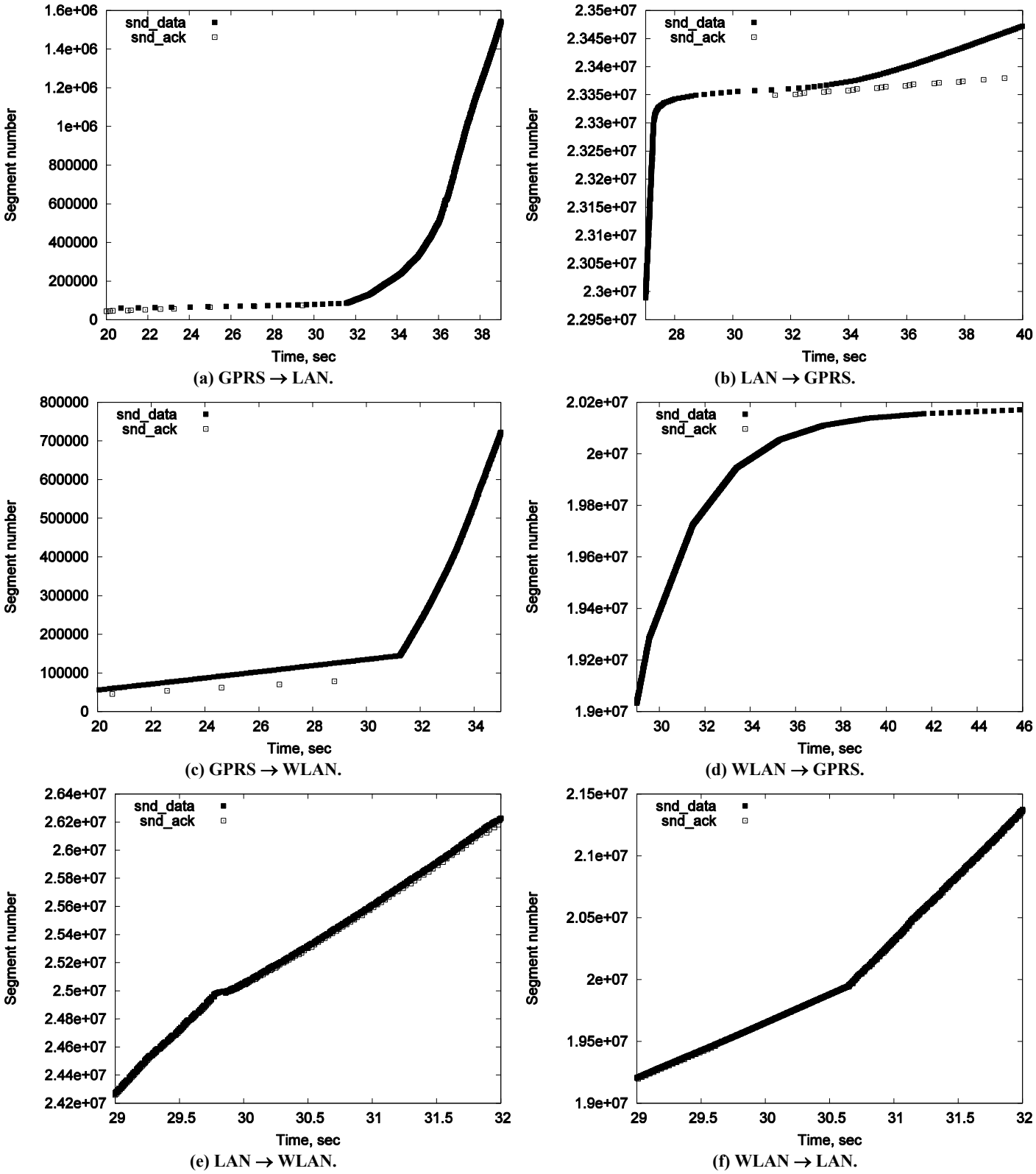
(c) GPRS → WLAN.

(d) WLAN → GPRS.

(e) LAN → WLAN.

(f) WLAN → LAN.

Figure 2. Measured behavior of a TCP flow during a vertical handover in the testbed.



(a) GPRS → LAN.

(b) LAN → GPRS.

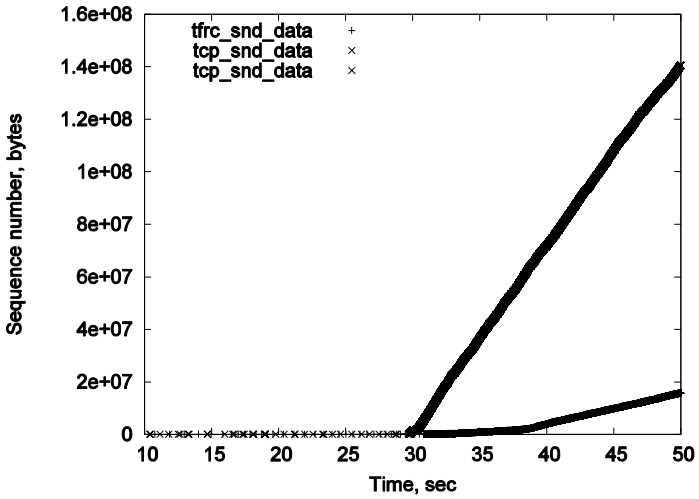
(c) GPRS → WLAN.

(d) WLAN → GPRS.

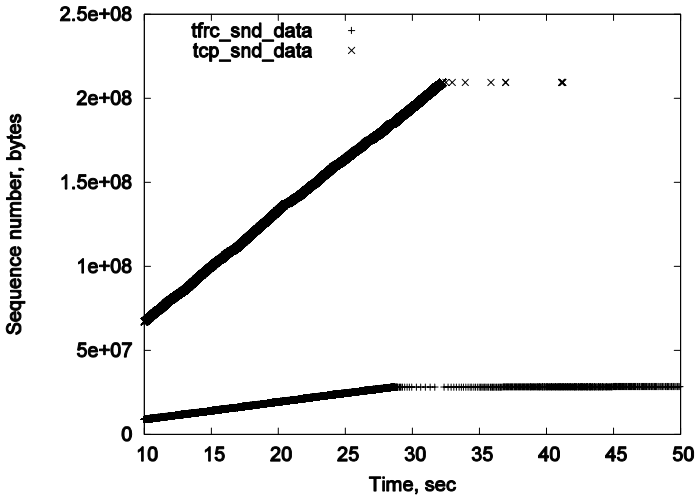
(e) LAN → WLAN.

(f) WLAN → LAN.

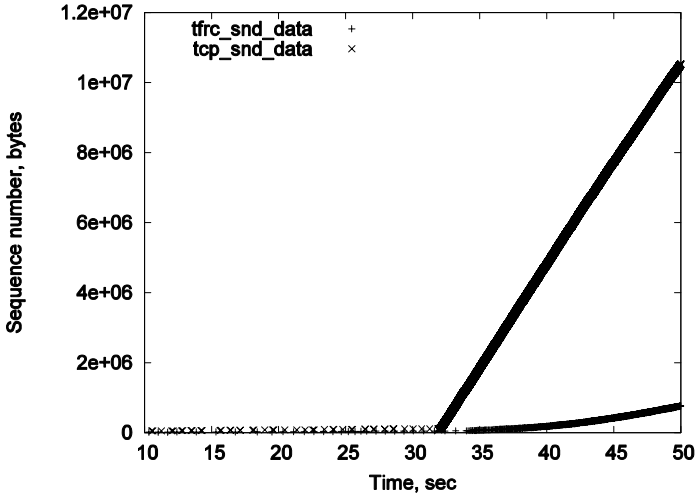
Figure 3. Measured behavior of a TFRC flow during a vertical handover in the testbed.



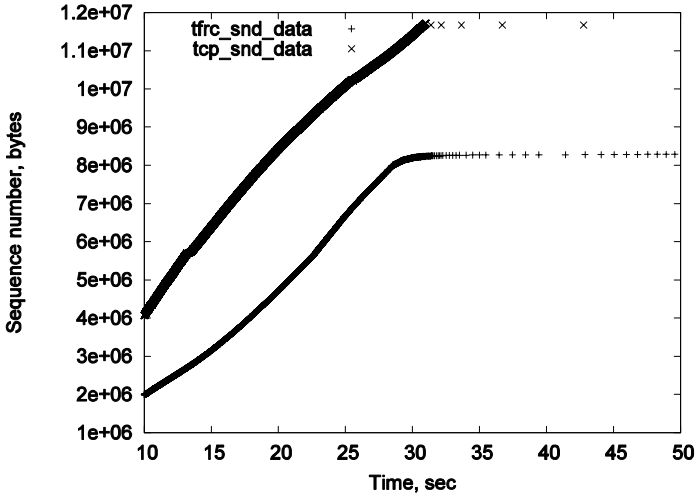
(a) GPRS → LAN.



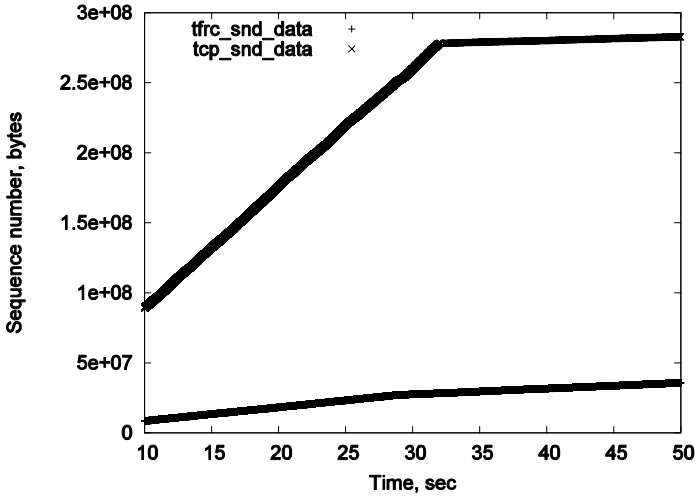
(b) LAN → GPRS.



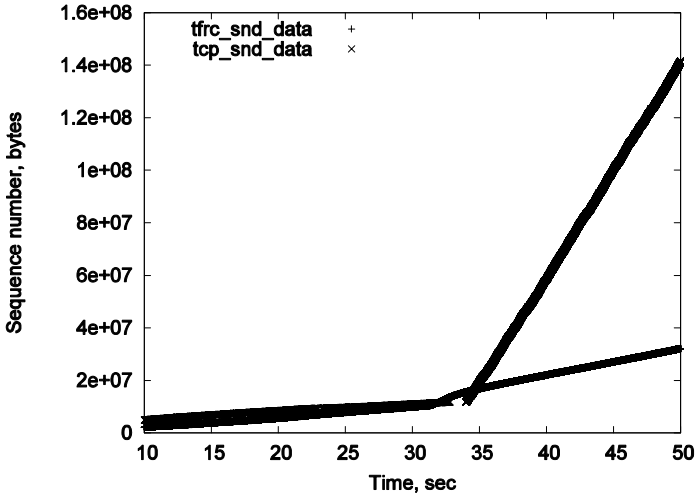
(c) GPRS → WLAN.



(d) WLAN → GPRS.

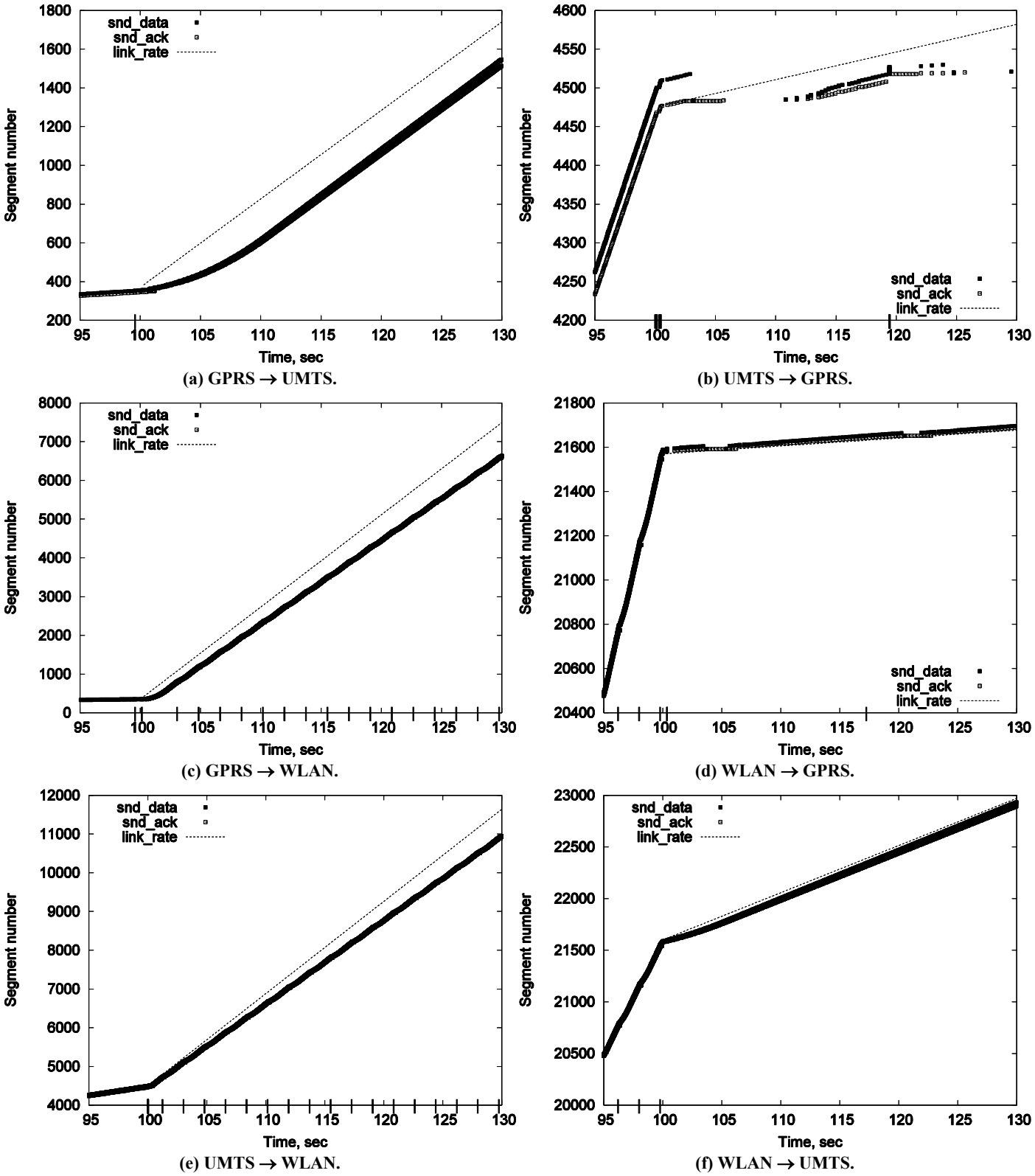


(e) LAN → WLAN.



(f) WLAN → LAN.

Figure 4. Measured behavior of a TFRC flow with a competing TCP flow during a vertical handover in the testbed. Handovers are triggered at the 30th second.



(a) GPRS → UMTS.

(b) UMTS → GPRS.

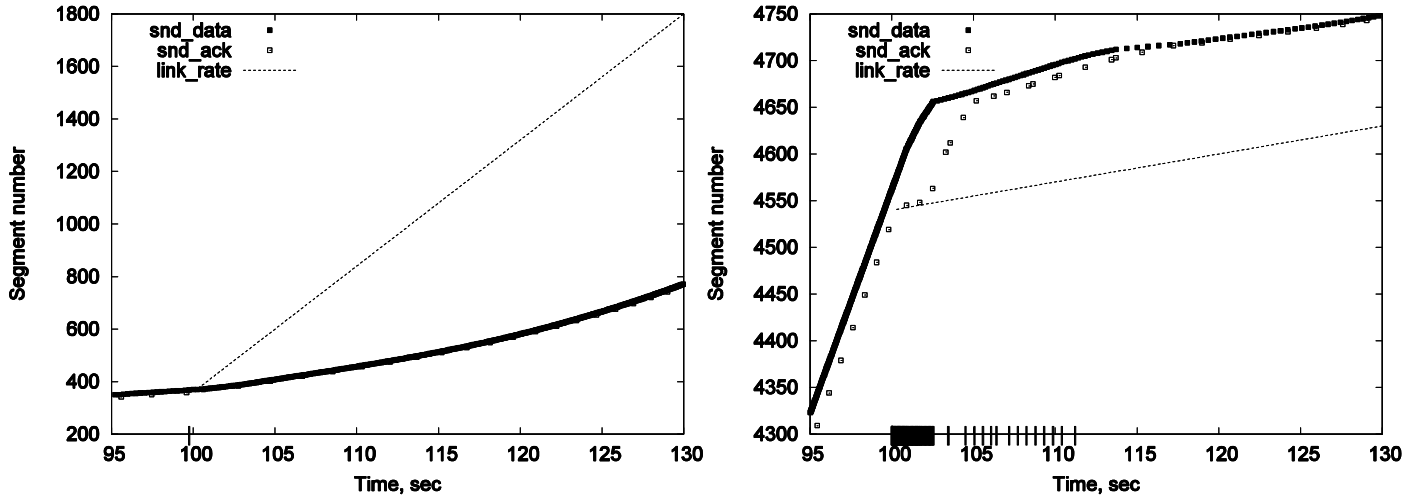
(c) GPRS → WLAN.

(d) WLAN → GPRS.

(e) UMTS → WLAN.

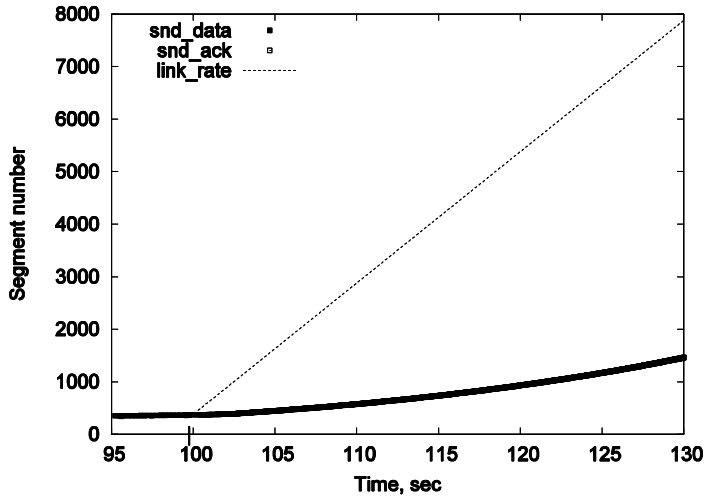
(f) WLAN → UMTS.

Figure 5. Simulated behavior of a TCP flow during an ideal handover. Marks on the x-axis show drops at the bottleneck queue.

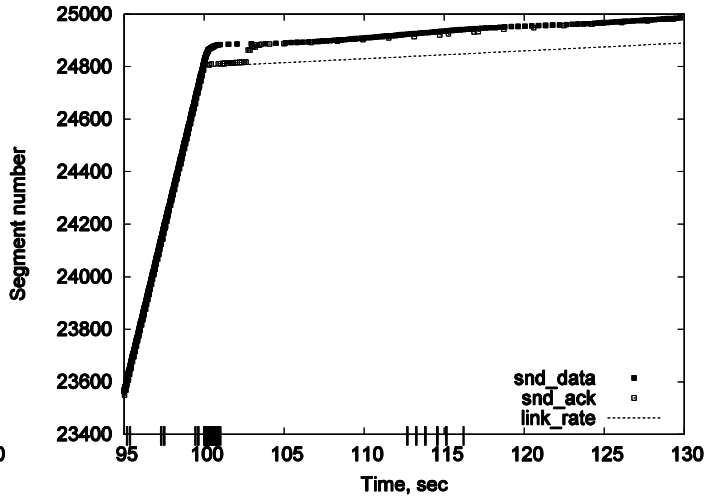


(a) GPRS → UMTS.

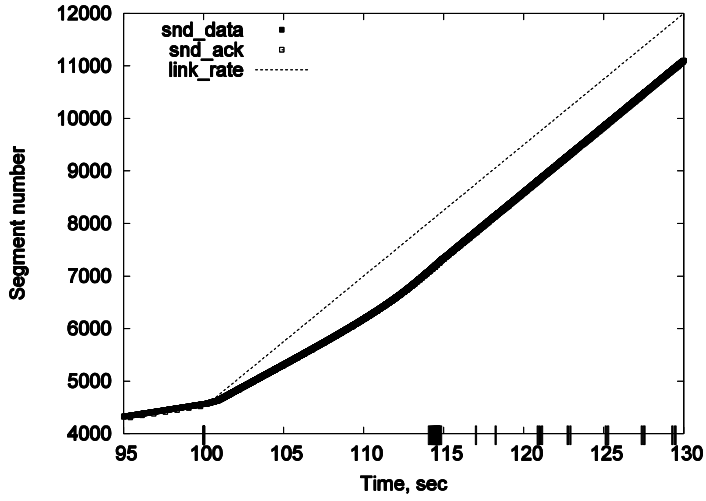
(b) UMTS → GPRS.



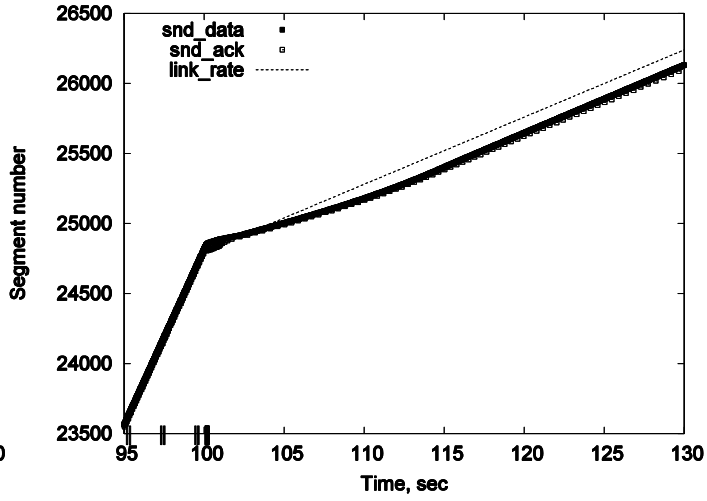
(c) GPRS → WLAN.



(d) WLAN → GPRS.

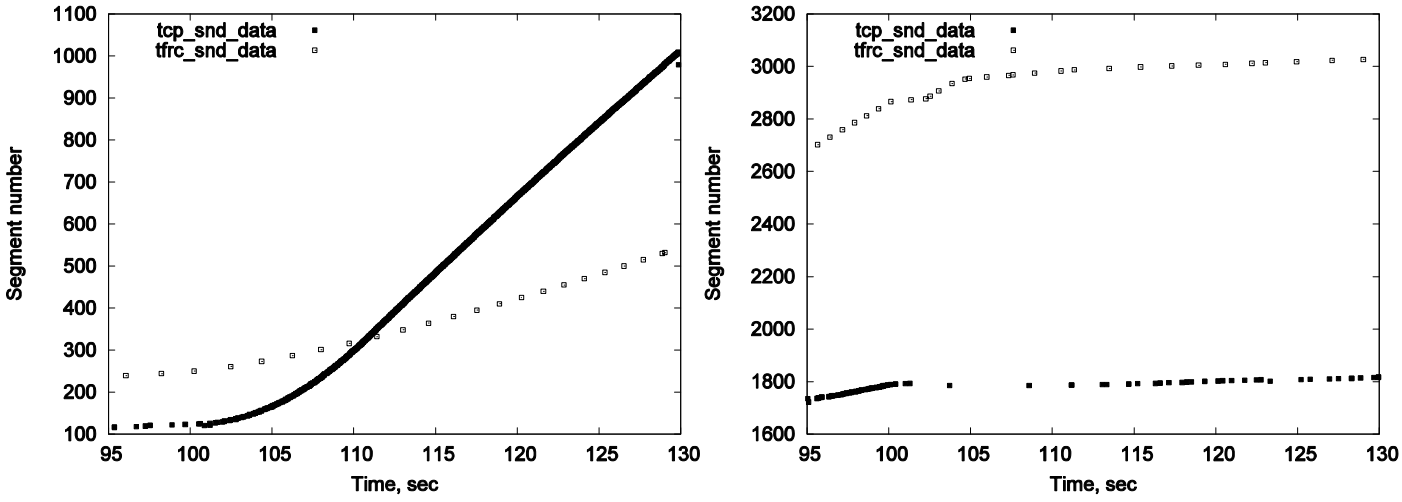


(e) UMTS → WLAN.



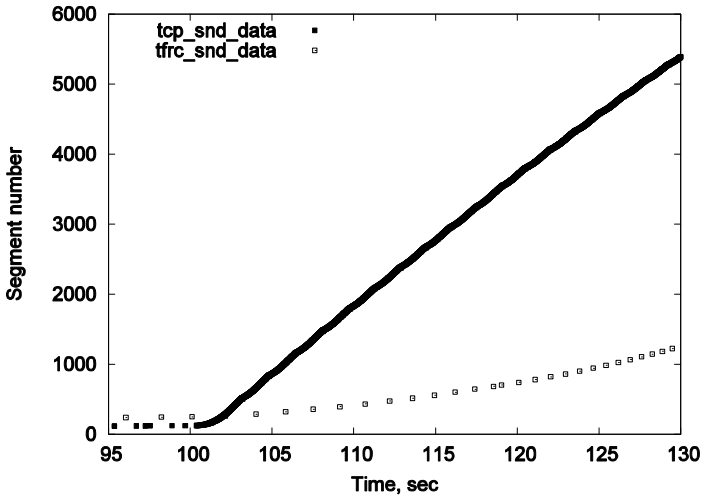
(f) WLAN → UMTS.

Figure 6. Simulated behavior of a TFRC flow during an ideal handover. Marks on the x-axis show drops at the bottleneck queue.

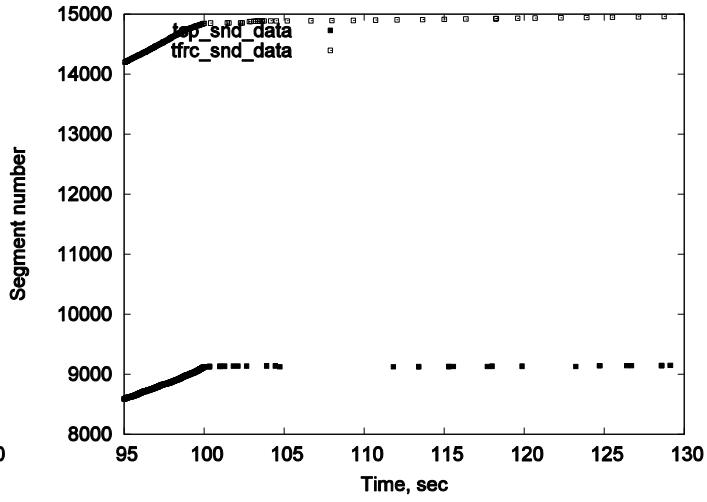


(a) GPRS → UMTS.

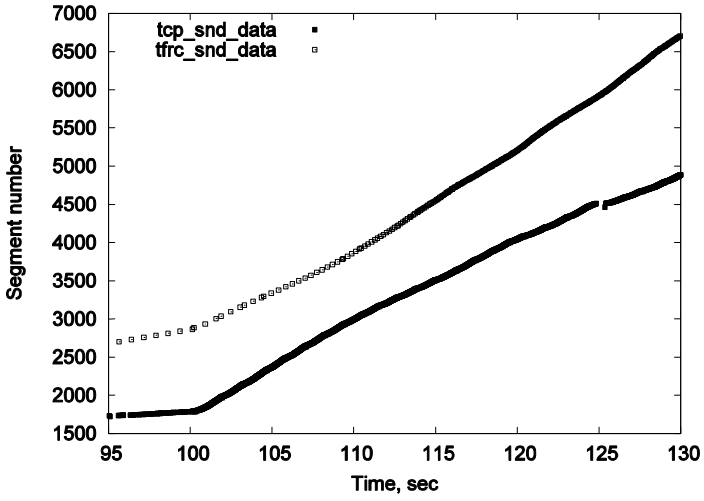
(b) UMTS → GPRS.



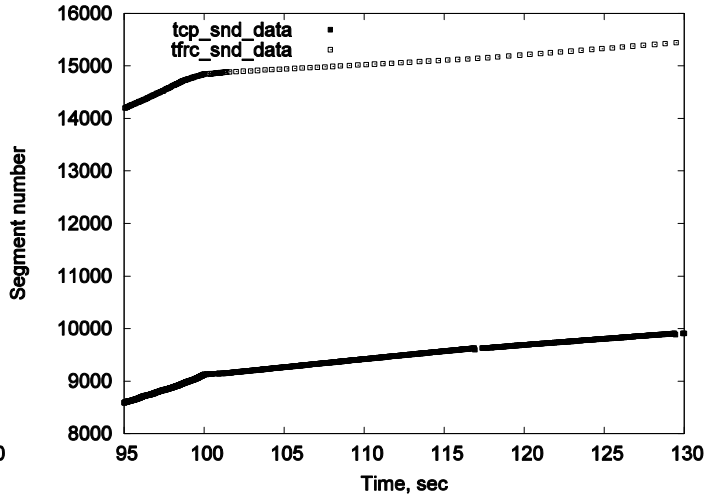
(c) GPRS → WLAN.



(d) WLAN → GPRS.



(e) UMTS → WLAN.



(f) WLAN → UMTS.

Figure 7. Simulated behavior of a TFRC flow with a competing TCP flow during a vertical handover. Handovers are triggered at the 100th second.

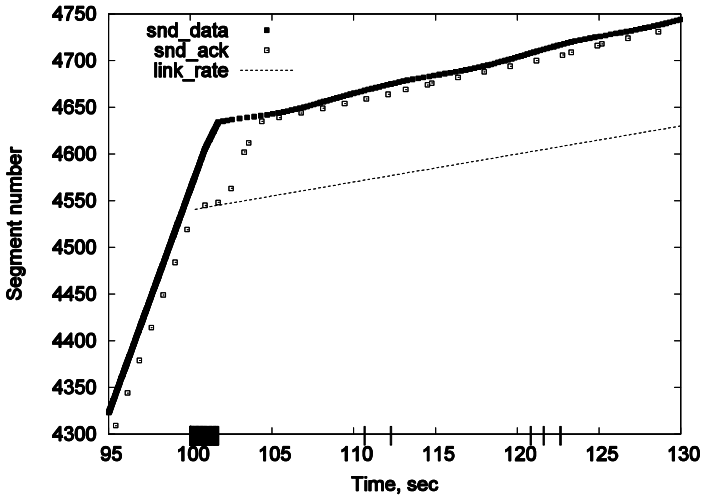


Figure 8. Effect of self clocking on TFRC (UMTS → GPRS).

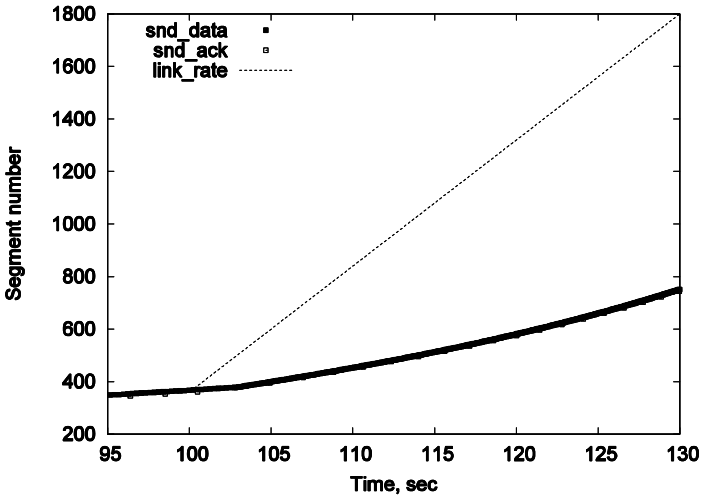
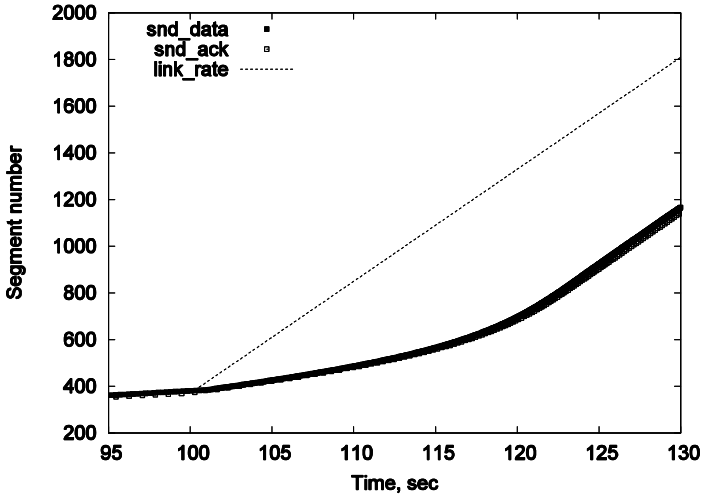
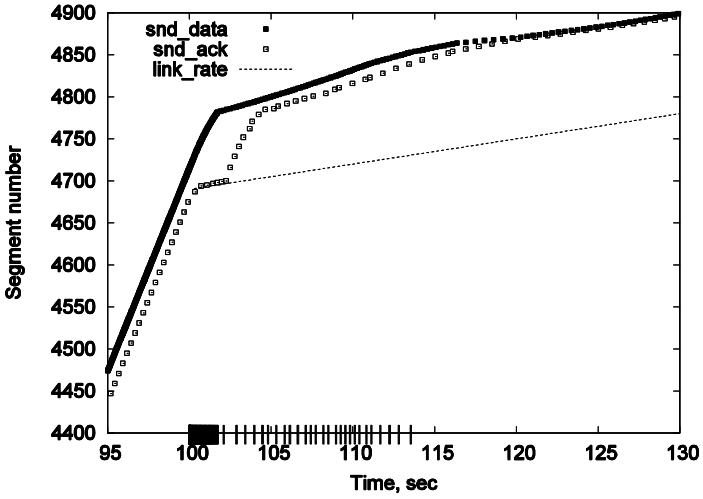


Figure 9. History discounting disabled in TFRC (GPRS → UMTS).

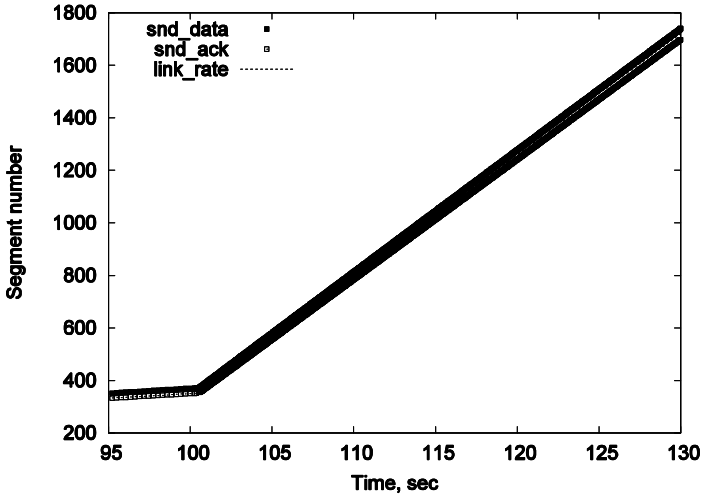


(a) GPRS → UMTS

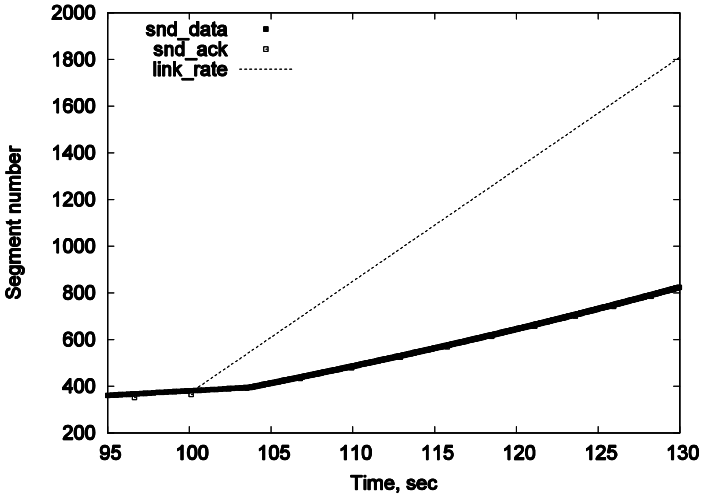


(b) UMTS → GPRS.

Figure 10. Higher feedback frequency in TFRC, three ACKs per RTT.

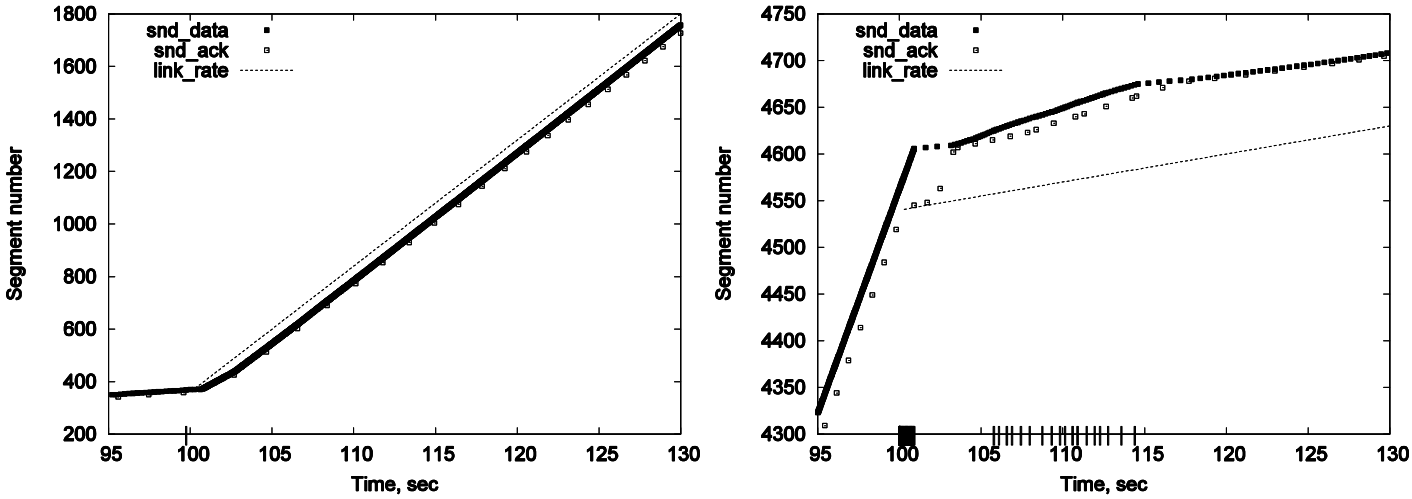


(a) TCP



(b) TFRC

Figure 11. Effect of overbuffering on TCP and TFRC (GPRS → UMTS).



(a) GPRS→UMTS (b) UMTS → GPRS

Figure 12. Explicit handover notification in TFRC.