University of Helsinki
Department of Computer Science
Series of Publications C, No. C-2012-2

Enhancing TCP with Cross-layer Notifications and Capacity Estimation in Heterogeneous Access Networks

Laila Daniel, Markku Kojo

Helsinki, February 8, 2012

Technical Report C-2012-2

University of Helsinki
Department of Computer Science
P. O. Box 68 (Gustaf Hällströmin katu 2b)
FIN-00014 University of Helsinki, FINLAND

Enhancing TCP with Cross-layer Notifications and Capacity Estimation in Heterogeneous Access Networks

Laila Daniel, Markku Kojo
Department of Computer Science, University of Helsinki
Technical Report C-2012-2
February 8, 2012
17 pages

Abstract. Finding the available bandwidth for a TCP connection is an important problem as it allows the connection to have high throughput, low packet loss rate and fairness. As flows arrive and depart, the state of a TCP connection at any point in time is very dynamic and it is difficult to find the accurate available capacity for a TCP connection quickly. This becomes extremely important in a heterogeneous access network environment where the path characteristics of a TCP connection may suddenly change due to a vertical handoff. We adapt the Packet-pair and PathChirp algorithms combined with cross-layer notifications regarding the access link bandwidth and delay to quickly determine a rough available bandwidth for the TCP connection. We apply our algorithms in simulation experiments covering various scenarios, including the beginning of a TCP connection, after a vertical handoff, and after a prolonged interruption in the connection. In vertical handoff scenarios we also apply the TCP algorithms we have earlier developed to mitigate the problems of TCP due to a vertical handoff. Results show that the proposed algorithms yield better throughput and reduction in packet losses compared to TCP.

1 Introduction

The Transmission Control Protocol (TCP) [28] is used by a number of applications in the Internet such as e-mail, Web browsing, file-transfer, streaming audio, video and P2P [36]. Recent measurement studies on Internet traffic show that TCP still dominates the *volume* of the Internet traffic (in terms of packets and bytes) whereas UDP now often accounts for the larger fraction of the total *number* of flows on a given link due to the rise in the streaming and P2P traffic [9].

Finding the available bandwidth of a TCP connection is an important problem as it allows the connection to have high throughput, low packet loss rate and fairness. As flows arrive and depart, the state of a TCP connection at any point in time is very dynamic and it is extremely difficult to find the available capacity for a TCP connection accurately. In this paper we consider three scenarios in which a TCP connection is in its 'initial' phase acquiring its bandwidth share. The three scenarios are, the beginning of the TCP connection, after a vertical handoff and after a disconnection period. In all these three cases the available bandwidth is unknown to the TCP connection as there is no feedback from the current path or the feedback is too old and should be considered stale.

TCP congestion control algorithms [5, 18] enable TCP to adapt to the characteristics of the end-to-end path. During the initial slow start phase of a TCP connection, TCP rapidly probes for the available bandwidth in an exponential manner. The slow start phase usually ends in a slow

start overshoot, thereby losing many packets and after recovery TCP moves to the congestion avoidance phase where it probes for the bandwidth in a linear way. If the end-to-end path had a high bandwidth-delay product (BDP), TCP slow start algorithm will take many round-trips ($\log_2$BDP) to reach the available capacity [18]. In the case of high BDP paths, it may be possible to have high packet losses due to slow start overshoot and retransmission timeouts are needed to recover the lost packets. Many research proposals exist to improve the start-up behaviour of TCP. Quick-Start is one of those approaches which sets the *cwnd* and *ssthresh* to an appropriate rate agreed by all the routers in the end-to-end [13, 32]. RAPID [21] and NF-TCP [6] are congestion control algorithms that use PathChirp [30] algorithm to find the available bandwidth of the end-to-end path continuously during the life-time of a connection. NF-TCP combines ECN [29] for congestion avoidance.

Internet access with a mobile device is extremely popular today. Contemporary mobile devices support multiple radio technologies such as Enhanced General Packet Radio Service (EGPRS) [34], Universal Mobile Telecommunication System/High Speed Down Link Packet Access (UMTS/HSDPA) [1], 3GPP Long Term Evolution (LTE) [2] and Wireless Local Area Network (WLAN) [17] to provide wireless access to the Internet. For instance, the current state-of-the-art mobile phone, Nokia N8-00, supports multiple radio interfaces such as GPRS/EDGE, HSDPA (maximum speed up to 10.2 Mbps), HSUPA (2.0 Mbps) and WLAN IEEE 802.11 b/g/n. It is envisaged that in the future mobile devices can support even ten or more radio interfaces to support a wide range of Internet applications using a broad range of wireless access technologies [31]. TCP may experience a variety of problems when a mobile device performs a vertical handoff which is the switching of an active connection from one access network to another having a different link layer technology [25]. Different link layer technologies have widely different link characteristics such as data rate and latency. As the sending rate of a TCP connection depends on the end-to-end path properties, significant changes in the link characteristics due to a vertical handoff affects TCP performance [11, 15]. The problems of TCP due to a vertical handoff have been studied and many solutions are proposed to mitigate them. After a vertical handoff, finding the capacity of the new path is similar to that of finding the initial available bandwidth. A solution to this problem is particularly important when handoff happens to a new path with higher BDP compared to the old path as TCP has the inherent problem of quickly achieving the available high capacity.

The third scenario where TCP determines its share of bandwidth is after a disconnection. The disconnection can be due to a link outage in wireless networks or due to a break-before make handoff where the connection to the old access router breaks before the connection to the new access router is made [25]. Usually if the disconnection time is greater than the TCP retransmission timeout (RTO), RTO timer expires and TCP goes to the slow start. If more than one timeout occur during the disconnection, the *ssthresh* usually will become quite low resulting in a low sending rate and underutilization as the TCP sender is forced to leave the slow start early. This scenario is important as most of the current handoff scenarios are break-before-make handoffs. Also intermittent connectivity is a frequent phenomenon in wireless network, achieving the appropriate bandwidth for a TCP connection is an important task after the connectivity is resumed.

In this report we propose a bandwidth estimation algorithm to estimate the available bandwidth of the end-to-end path of a TCP connection in the above three scenarios. The proposed bandwidth estimation algorithm is mainly based on the Packet-pair algorithm [19] to estimate the bottleneck capacity and the PathChirp algorithm [30] to estimate the available bandwidth of the end-to-end

path of a TCP connection. We use the PathChirp algorithm as many studies have shown that PathChirp estimates available bandwidth with good accuracy [21, 35]. We speed up the bandwidth estimation with the additional information from the cross-layer notifications such as the bandwidth and delay of the access links and the occurrence of a handoff or disconnection. This helps the TCP connection to achieve its share quickly and also to avoid the problems of slow start overshoot.

In our papers [11, 12] we developed algorithms to mitigate the problems of TCP due to a vertical handoff. We use cross-layer notifications regarding the access link characteristics involved in a handoff to determine the proper values for various parameters such as *cwnd*, *ssthresh* and retransmission timeout (RTO)value. The results of extensive simulation study show that our algorithms are useful in the vertical handoff scenarios. These algorithms can be used after a disconnection period to combat the problems of TCP arising in this scenario. In the handoff scenarios to a higher bandwidth/higher BDP path compared to the old path is challenging as TCP has inherent problems in adapting to a high capacity available. The proposed bandwidth estimation algorithms and the vertical handoff algorithms are used both after a vertical handoff and also after a disconnection.

We have implemented our algorithm in ns-2 [26] and performed experiments to evaluate the effectiveness of the proposed scheme. We use TCP with Quick-Start algorithm [32] as a benchmark to evaluate the performance of our proposed algorithm. Quick-Start algorithm allows TCP to have optimal performance as it 'knows' a TCP connection's share of bandwidth but in practice Quick-start algorithm finds deployment problems as it requires support by all the routers on the end-to-end path. Our results show that the proposed bandwidth estimation improves the TCP start-up performance. Vertical handoff algorithms combined with bandwidth estimation algorithms are effective especially in handoffs to high-bandwidth/high-BDP access links.

Section 2 presents the related work in this area and Section 3 describes the proposed algorithm used and briefly describes the vertical handoff algorithms. Section 4 presents the simulation results and Section 5 gives the conclusions of the paper and outlines the future work.

2 Related Work

Many research proposals exist to improve the start-up behaviour of TCP. The foremost one is due to Hoe [16] in which the *ssthresh* is advised to be set to the BDP of the end-to-end path. *Quick-Start* algorithm [13, 32] sets the initial congestion window to a higher value than the default if all routers along the communication path approve the Quick-Start request for higher sending rate. The Quick-Start request is included in a hop-by-hop IP option with the TCP SYN packet. Quick-Start can also be applied after a vertical handoff as proposed in [33]. Even though Quick-Start allows any TCP connection to start with an accurate share of the bandwidth for that connection, it requires all the routers on the path to implement the Quick-Start algorithm which makes the deployment of Quick-Start rather hard.

The bandwidth estimation methods can be used to aid a TCP flow in estimating its share during its lifetime. Recent research has shown that the bandwidth estimation tools proposed for wired networks are not suitable for wireless networks due to the special characteristics of wireless networks such as rate adaptation, contending traffic, loss recovery using ARQ mechanisms and scheduling

etc. ProbeGap [23] and WBest [24] are estimation tools specially designed for wireless LAN environments. ProbeGap [23] needs an a priori knowledge of the initial capacity [24]. WBest [24] can estimate the available bandwidth accurately only in controlled WLAN environments and is not effective in cellular environments [22]. As many bandwidth estimation tools including ProbeGap and WBest inject probe packets to the network to estimate the available bandwidth, use of separate probe packets will overload the bottleneck link of the network path.

RAPID [21] is a rate-based end-to-end transport protocol, the first proposal which uses *chirping*, i.e., sending packets as a stream in which the sending rates of the packets are exponentially increasing, for end-to-end congestion control. In RAPID, bandwidth estimation is based on PathChirp algorithm and it sends chirps of packets in a stream such that the average rate of the chirps in a stream is the rate estimated in the previous stream. As the chirps are placed at an exponential rate, the authors state that a connection using RAPID can reach a Gigabit capacity path in four round-trip times. As RAPID's available network capacity estimation fully relies on PathChirp algorithm which may not be able to give accurate bandwidth estimation in wireless networks, the usefulness of RAPID in wireless networks may be limited.

NF-TCP [6], a network-friendly TCP for delay insensitive applications uses chirping congestion control. It combines the congestion control with ECN-based congestion avoidance techniques. NF-TCP uses additions probe packets to estimate the available bandwidth. It is not advisable to use additional probe packets in wireless networks which are bandwidth limited.

Setting the *cwnd* appropriately after a handoff is crucial both in avoiding the congestion-related losses due to a handoff to a lower bandwidth-delay product (BDP) link as well as in effectively utilizing the higher BDP of a new link after a handoff. Slow starting [20], using Quick-Start [33], setting *cwnd* using the cross-layer notifications on the access link bandwidth and delay [11] and using packet pairs [37] to find the capacity are some of the proposals to effectively utilize the capacity after a handoff. A comprehensive survey on the proposals to set the *cwnd* after a handoff is given in [11].

3 Proposed available capacity algorithm

The available capacity estimation algorithm we propose in this paper consists of two phases. In the first phase, the bottleneck capacity estimation is carried out by sending TCP packets back-to-back. Using this bottleneck capacity, the available network capacity or available bandwidth of the connection is estimated by sending TCP packets at an exponentially increasing rate. Even though the basic ideas used in our algorithms are similar to the well-known Packet-Pair [19] and PathChirp [30], both algorithms are altered to leverage additional information from cross-layer notifications.

The algorithm is invoked when a notification from the Mobile node (MN) arrives at the TCP sender at beginning of a TCP connection or after a handoff. The notification includes the bandwidth and delay of the access link and the number of flows from the TCP sender to MN. Figure 1 gives the bandwidth estimation algorithm.

Algorithm is invoked when a notification from the Mobile node (MN) arrives at the TCP sender at beginning of a TCP connection or after a handoff. The notification includes the bandwidth and delay of the access link and the number of flows from the TCP sender to MN

Calculate the Bandwidth Delay product (BDP) of the access link
using the bandwidth and delay values in the cross-layer notification
$BDP_{link} = Bandwidth_{link} * Delay_{link}/MSS$

Bottleneck capacity estimation algorithm

$pkt\_count = min(4, BDP_{link}/(\#flows))$
If $pkt\_count \leq 1$ Continue with regular TCP behaviour
Else, TCP sender sends pkt_count of TCP packets back-to-back
TCP receiver calculates bottleneck bandwidth, bbw, sends it to TCP sender
$bbw = MSS/(p_2 - p_1)$
where p_1, p_2 the arrival times of the first and the second packet in a pair
If $(bbw > BDP_{link}), bbw \leftarrow BDP_{link}$

Available bandwidth (abest) algorithm - in general

TCP sender sends pkt_count of TCP packets as chirps.
The $lowrate$ and $highrate$ are lowest and highest rates
of the chirps and the spreadfactor is sf
$sf = exp(log(highrate/lowrate)/pkt\_count)$
TCP receiver calculates the available bandwidth, $abest$, sends it to TCP sender
$ssthresh = abest * RTT/MSS$
$(abest > bbw)$, $abest \leftarrow bbw$
The estimation at any abest cycle stops when
$abest < highrate$ or $2 * cwnd > ssthresh$

Abest cycle 1

$pkt\_count = min(cwnd, bbw * RTT/(\#flows * MSS))$
$lowrate = bbw/(\#flows * 2)$
$highrate = bbw/(\#flows)$
Estimated available bandwidth $abest_1$
$ssthresh = abest_1 * RTT/MSS$

Abest cycle 2

$pkt\_count = min(cwnd, abest_1 * RTT/(MSS))$
$lowrate = 2/RTT$
$highrate = abest_1 * 2$
Estimated available bandwidth $abest_2$
$ssthresh = abest_2 * RTT/MSS$

$\vdots$

Abest cycle n (n from 3 onwards)

$pkt\_count = min(cwnd, abest_n * RTT/(MSS))$
$lowrate = lowrate_{n-1} * 2$
$highrate = highrate_{n-1} * 2$
Estimated available bandwidth $abest_n$
$ssthresh = abest_n * RTT/MSS$
Repeat the abest cycle until
$abest < highrate$ or $2 * cwnd > ssthresh$

Figure 1: Bandwidth estimation algorithm

Bottleneck capacity estimation

The bottleneck capacity is estimated using packet dispersion technique that measures the end-to-end capacity of a network path [19]. The packet dispersion technique sends two or more packets back-to-back into the network. After traversing the bottleneck link, the time dispersion between the two packets is linearly related to the bottleneck link capacity. We assume that the access link forms the bottleneck of the end-to-end path so the bandwidth delay product (BDP) of the access link determines the maximum number of packets that can be send in a window. The algorithm calculates the BDP of the access link from the values of the bandwidth (data rate) and propagation delay of the access link obtained from the cross-layer notification. Since the cross-layer notification gives the information on the number of flows from the TCP sender to the MN, we set the upper bound for the share of the capacity for each flow as the BDP divided by the number of flows. At the beginning of a TCP connection, the initial window of the connection can be a maximum of 4 packets [4]. So the bottleneck capacity estimation cycle consists of sending the minimum $(4, BDP_{link}/(\#flows * MSS))$ TCP packets back-to-back. The TCP receiver estimates the bottleneck capacity as $bbw = MSS/(p_2 - p_1)$ where p_1 and p_2 are the arrival times of the first and the second packet in a pair and send this information to the TCP sender. The capacity estimation algorithm allows us to confirm whether the access link is the bottleneck of the end-to-end path. If *bbw* estimated is turned out to be greater than BDP_{link}, *bbw* is set to BDP_{link} assuming that the notification gives us more accurate value than the bandwidth estimate using 2 or 3 packet-pairs.

If the $BDP_{link}/(\#flows))$ is less than 2, the algorithm sets the initial *cwnd* as 1 and follows the baseline TCP behaviour. This helps to avoid the aggressive behaviour leading to packet losses by setting an initial window greater than 2.

Available bandwidth estimation (abest) cycles

Once the bottleneck bandwidth is obtained, the next phase is the available bandwidth estimation (abest) cycle. The available bandwidth estimation (abest) algorithm to estimate the available bandwidth is based on the PathChirp algorithm [30]. Here TCP packets are sent at an exponentially increasing data rate with a spreadfactor which is equal to the ratio of the data rate of two adjacent packets. The basic idea of the PathChirp algorithm is that if the data rate is less than the available bandwidth the queuing delay experienced by a packet will be zero and if the data rate is greater than the available bandwidth the packet will experience a queuing delay. As the TCP sender sends at an increasing rate there will be an increasing trend in the queueing delay at the receiver once the sending rate becomes larger than the available bandwidth. It is possible that due to a short transient congestion, the queueing delay can increase. The PathChirp algorithm is taking care of this situation and checks if queueing delay comes back to zero after a short time. The PathChirp algorithm searches for a signature (to check the queueing delay increase is due to a transient overload). In our algorithm, as we are estimating the available bandwidth to calculate *ssthresh*, a limit to avoid slow start overshoot, the TCP receiver sets the highest value for the available bandwidth as the rate at which the queueing delay starts increasing and calculates the available bandwidth as in the PathChirp algorithm. This way we get a conservative bound to the *ssthresh* value.

The abest algorithm at the TCP sender sends chirps at an exponential rate. In any cycle, the number of chirps denoted by *pkt_count* is the minimum(*cwnd*, share of the capacity for a flow/MSS). The TCP sender initiates one or more abest cycles with different high and low rates by sending *pkt_count* of TCP packets with a spreadfactor equal to $exp(log(highrate/lowrate)/pkt\_count)$. The *highrate* and the *lowrate* are the highest data rate and the lowest data rate of the chirps in an abest cycle. The number of packets in a abest cycle is upper bounded by the current *cwnd*. So the growth rate of the window is similar to that of regular TCP algorithm. The TCP receiver estimates the available bandwidth and returns it to the sender. Once the estimate for the available bandwidth has been elicited is calculated *ssthresh* is set to the product of available bandwidth and RTT worth of packets. The abest cycle is exited if twice the *cwnd* is greater than *ssthresh* value set based on the available bandwidth. This allows the sender window to grow exponentially to the *ssthresh* quickly. The algorithm also exits the abest cycle if the available bandwidth estimated is less than the *highrate* of the chirp. If either of the above two conditions are not met, another abest cycle is initiated with new high and low rates and with a new spreadfactor. As a conservative approach, in any abest cycle, if the available bandwidth obtained is greater than the *bbw*, it will be set to *bbw*. The *cwnd* is incremented as in slow start phase of a TCP connection. Next we describe the abest cycles in detail.

Abest cycle 1

This is the first abest cycle after the capacity estimation cycle that estimates the bottleneck bandwidth *bbw*. Here the *pkt_count* is the $minimum(cwnd, bbw * RTT/(\#flows * MSS))$. The *cwnd* at the end of the capacity estimation cycle will be from 6-8 as the *cwnd* is incremented as in slow start. The highrate for the chirp cycle is of $(bbw/(\#flows))$ and the lowrate is the half of the highrate, i.e., $(bbw/2 * \#flows)$. The TCP receiver checks the difference in queueing delay of two packets and see whether there is an increase in queueing delay compared to the previous packets. If so, it set the rate of other high data rate chirps to the data rate of the packet where the increase in queuing delay is first noticed. The receiver calculates the average rate and send back this value to the TCP sender as $abest_1$ (available bandwidth of the abest cycle 1). The *ssthresh* is set to the product of abest and RTT in packets. RTT is calculated using the timestamps. If $abest_1$ is greater than or equal to the highrate and $2 * cwnd$ is smaller than *ssthresh*, the algorithm continues to the second abest cycle.

Abest cycle 2

In the second abest cycle, the highrate for the cycle is of $abest_1$ and the lowrate is $2/RTT$. The lowrate is taken the above value so that the chirps will we received in one RTT. The *pkt_count* will be the $minimum(cwnd, abest_1 * RTT/MSS)$. The *cwnd* for this cycle will range from 12-16 packets. The TCP receiver calculates the $abest_2$ and sends to the TCP sender. The ssthresh is set to the product of $abest_2$ and RTT. If $abest_2$ is greater than or equal to the highrate and $2 * cwnd$ is smaller than *ssthresh*, the algorithm continues to the third cycle.

Abest cycle n, $n \geq 3$

In the third abest cycle and the higher abest cycles, the highrate is twice the highrate of the previous cycle and the lowrate is twice the lowrate of the previous cycle. As the available bandwidth estimate of the previous cycle is greater or equal to the highrate of that cycle, in the new cycle the algorithm tries to see whether the available bandwidth is still higher than the highrate of the previous cycle. The high rates are probed to see if some flows have left the end-to-end path. As the lowrate is also doubled and the current *cwnd* is high, the spreadfactor will be low resulting in closer spaced chirps leading to a more accurate estimate. The abest cycles continue until one of the conditions for exiting for the abest cycle is met.

Once the available bandwidth is estimated, the *ssthresh* is finally set to the product of the available bandwidth and the RTT. The slow start overshoot that causes a large packet loss is avoided in this way. The *cwnd* is incremented as in slow start phase of a TCP connection.

3.1 VHO algorithms

In this section, we a briefly describe the algorithms to mitigate the problems of TCP due to a vertical handoff. As a vertical handoff results in abrupt significant changes in the bottleneck link characteristics, TCP may take several RTTs to adapt to the path changes during which TCP may experience problems such as occurrence of spurious RTOs, packet reordering, packet losses and unused connection time that affect the application performance.

In a vertical handoff, spurious RTOs may occur when a make-before-break handoff takes place from a low-delay link to a high-delay link. After the handoff, the ACKs will be delayed due to the high latency of the new high-delay link. Due to the small RTO value calculated on the basis of the old path, the TCP retransmission timer may expire before the arrival of the ACKs through the new link resulting in unnecessary retransmissions and *cwnd* reduction. Using the bandwidth and delay values of the old and the new access links, we find the traversal time for a *Data-ACK pair* at the time of handoff and if it is greater than the current RTO, there is a possibility of the occurrence of spurious RTOs. The algorithm calculates the minimum RTO, *minrto*, based on the new access link delay and update the RTO timer immediately after the handoff thereby avoiding the occurrence of spurious RTOs.

Congestion-related packet losses may occur in a vertical handoff if the new path after a handoff has less capacity than the old path. As the access links are most often the bottleneck links, packet losses due to congestion may occur if the *FlightSize* at the time of handoff is greater than the buffering capacity of the new link. If this condition is met, the algorithm sets the *cwnd* and *ssthresh* to the BDP of the new access link.

Packet reordering occurs when a make-before-break handoff takes place from a high-delay link to a low-delay link. The packets with high sequence numbers sent after the handoff through the new link arrive at the TCP receiver earlier than the packets sent before the handoff through the old link resulting in packet reordering. As a result of reordering, the TCP receiver sends duplicate acknowledgements (*dupacks*) over the new link and when the TCP sender gets three *dupacks*, it triggers a false *fast retransmit* causing many unnecessary retransmissions and reduction in *cwnd* and *ssthresh*. The idea behind the algorithm to combat the problems due to packet reordering is to determine how likely packet reordering occurs given the bandwidth of the two access links and

use this information along with Duplicate Selective acknowledgements (DSACKs) [14], an advanced feature of TCP, for detecting unnecessary retransmissions.

A break-before-make handoff is likely to result in unused connection time. If the disconnection time is more than the RTO value of a TCP connection, the RTO timer may expire several times during the disconnection period, each time doubling the RTO value [27]. When the connectivity is resumed, the TCP sender needs to wait until the RTO timer expires again before attempting another retransmission. This unused connection time and the repeated reduction in *ssthresh* are the major problems of TCP arising from the disconnection caused by a break-before-make handoff. In our algorithm to reduce the unused connection time, the TCP sender immediately retransmits the first unacknowledged packet if it is already in RTO recovery when the handoff notification arrives. Also the *ssthresh* variable is set to the BDP of the new link to avoid the repeated reduction in *ssthresh*.

The problems of TCP in a vertical handoff and the cross-layer assisted algorithms that we have proposed to mitigate those problems are described in detail in our papers [11, 12].

4 Evaluation of the proposed algorithms

We evaluate the proposed algorithms in three scenarios, namely, the initial phase of a TCP connection, after a vertical handoff and after a disconnection. We use the ns-2 simulator to model these scenarios and evaluate the proposed algorithms.

In the current study, we model only access networks roughly resembling wireless networks such as EGPRS, UMTS/HSDPA, LTE and WLAN to connect the mobile node (MN) to the Internet. The two wireless access links involved in the handoff are represented by their bandwidth and the propagation delay, the two characteristics that affect the TCP behaviour. We often refer to the propagation delay of an access link as the 'delay' of the link. We use the notation ' x/y link ' to denote a link of bandwidth x and delay y. The bandwidth and delay of the wireless access networks used in our experiments are given in Table 1.

Table 1: Data rate and Propagation delay of the Wireless Access Networks used

Access Network	Data rate	Propagation Delay (one way)
EGPRS	200 Kbps	300 ms
HSDPA	2 Mbps, 6 Mbps	50 ms
LTE	50 Mbps	15 ms
WLAN	54 Mbps	2 ms, 4 ms

In this paper we are carrying out a comparative study of our proposed algorithms with TCP versions that basically follow the congestion control algorithms described in RFC 5681 [5]. The versions of TCP we evaluate in this simulation experiment are named as follows: Baseline TCP, Qs, Iw10, Vho, AbestVho.

In our experiments, we use TCP SACK [7] as the baseline TCP to evaluate its performance in a vertical handoff and to quantify the improvements in TCP performance due to the use of our

algorithms. We refer to TCP SACK as regular TCP or just 'TCP' in presenting our experimental results. The TCP initial window of three 1460-byte segments is selected based on RFC 3390 [4]. Delayed ACK [8], Limited transmit [3] and DSACK [14] are enabled in TCP. The receiver advertised window is set to 5000 packets so that it will not be a limiting factor in setting the congestion window. The TCP packet size is 1500 bytes inclusive of the TCP/IP headers. The router buffer size of each link is set to the BDP of the access link with at least five packets as the minimum value and no link capacity allocation delay is modelled.

The QS version of TCP is the baseline TCP with Quick-start [13,32,33] applied both at the start of a TCP connection and after a vertical handoff. We use QS version as a benchmark to evaluate the performance of our proposed algorithm. Quick-Start algorithm allows TCP to have optimal performance as it 'knows' a TCP connection's share of bandwidth but in practice Quick-start algorithm finds deployment problems as it requires support by all the routers on the end-to-end path.

The Iw10 [10] version is the baseline TCP with initial window of 10 packets.

The Vho version is the baseline TCP with the vertical handoff algorithms we have proposed in [11, 12]. Vho TCP sets the initial *ssthresh* to the product of the access link bandwidth and delay obtained in the cross-layer notification.

The AbestVho is the baseline TCP with available bandwidth estimation algorithms proposed in this paper with the Vho algorithms. In all the experimental scenarios described in this paper AbestVho evaluates the available bandwidth within in three RTTs.

In our experiments, only one mobile node and one correspondent node are considered. One to four bulk TCP flows are transferred from the correspondent node to the mobile node. The small number of multiple flows in the study is adequate considering that a mobile device typically runs a small number of applications simultaneously.

The cross-layer notification is delivered to the TCP sender which is set at the beginning of a connection, after the occurrence of a vertical handoff or a disconnection and is modelled using a user-defined command in ns-2 with the link characteristics as its parameters.

4.1 Initial phase of a TCP connection

In the initial phase of a TCP connection, TCP enters the slow start and probes the available bandwidth for the connection. Initially the slow start threshold (*ssthresh*) is set to a very high value and the congestion window (*cwnd*) is set to a small value determined by the initial window and TCP increments the *cwnd* by one MSS for every incoming ACK till a packet loss occurs. In slow start the *cwnd* grows exponentially and when the slow start ends, a large number of packets may be dropped especially in the case of high bandwidth-delay product networks. This is called s slow-start overshoot which may need an RTO to recover the lost packets. Subsequently TCP will enter the congestion avoidance phase with a smaller *cwnd* thereby reducing the sending rate drastically.

In this experiment of studying the behaviour of TCP during the initial phase, we sent packets 10, 100 up to 400 packets and find the goodput of the connection. Goodput is defined as the number of useful data bytes transmitted which excludes the retransmitted data packets.

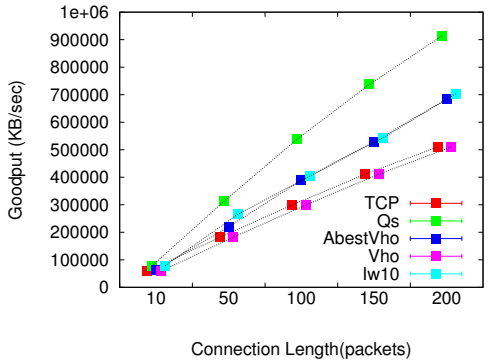

Figure 2: Connection start-up behaviour - Bottleneck link 50000 Kbps/15 ms LTE link - 1 Flow

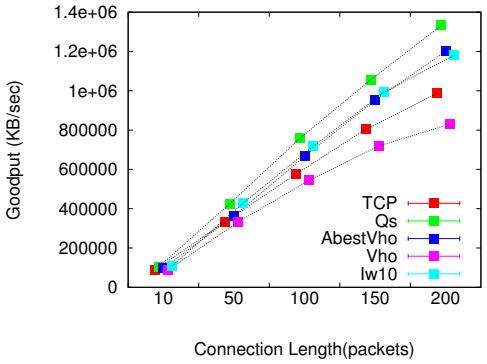

Figure 3: Connection start-up behaviour . Bottleneck link 54000 Kbps/2 ms WLAN link - 1 Flow

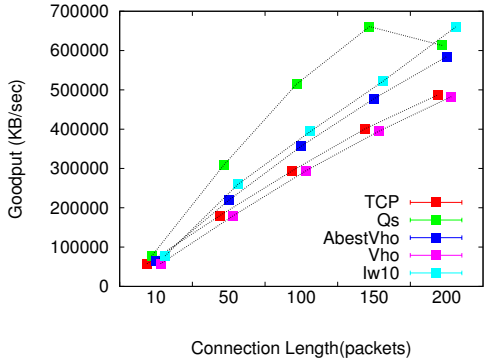

Figure 4: Connection start-up behaviour - Bottleneck link 50000 Kbps/15 ms LTE link - 4 Flows

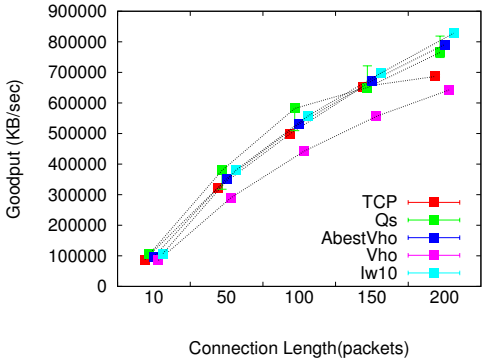

Figure 5: Connection Start-up behaviour Bottleneck link 54000 Kbps/2 ms WLAN link - 4 Flows

The wireless access links we evaluated here are the high bandwidth links such as LTE and WLAN. LTE (50 Mbps/15 ms) has a BDP of 125 packets. The WLAN (54 Mbps/ 2 ms) is a very fast link with BDP of 18 packets. We have selected the above access links as TCP has difficulty in quickly adapting to the high bandwidth/BDP available.

Figures 2 and 4 show the goodput for one and four TCP flows with different number of packets are sent through an LTE link using one and four simultaneous TCP connections respectively. In all the three cases QS TCP achieves the highest throughput. AbestVho and Iw10 perform comparably achieving higher throughput than TCP. Iw10 is aggressive and packet losses occur for Iw10 in all the three cases whereas AbestVho has zero packet loss. As can be seen in Figure 4, AbestVho comes closer to the goodput achieved by QS TCP.

Figures 3 and 5 show the goodput for one and four TCP flows with different number of packets are sent through a WLAN link. AbestVho has performs comparable to QS TCP in the case of one flow. As Quick-Start has problems in setting the initial rate for multiple TCP flows when the BDP is low. Iw10 achieves similar goodput than AbestTCP in the four TCP flows but it is aggressive compared to AbestVho.

It is interesting to note that Vho, which sets the *ssthresh* to the BDP of the access link achieves a

smaller goodput than TCP. This is because the end-to-end BDP is always higher than the BDP of the access link and by setting *ssthresh* to the BDP of the access link limits the exponential increase of the *cwnd* and thereby the sending rate in the slow start phase.

The results of the above experiments show that AbestVho is less aggressive and utilizes the high bandwidth/high BDP links efficiently.

4.2 After a Vertical handoff (Vho)

In our experiments a handoff can occur once in the lifetime of a TCP connection in any of the TCP phases, namely, slow start, slow start overshoot, fast retransmit/fast recovery and congestion avoidance.

In order to study the behaviour of TCP with vertical handoffs we focus on the TCP behaviour immediately after a handoff. As a performance metric, we calculate the time taken to transfer (to get the acknowledgment) n packets through the new link after the handoff. This metric is calculated with respect to the slowest flow when more than one flow is present.

We have chosen the access links EGPRS, HSDPA, WLAN and LTE to show the effectiveness of AbestVho in a vertical handoff. We experiment both the make-before-break and break-before-make handoffs [25]. In each of the experiments we study the behaviour for n long TCP flows where n equals 1, 2 and 4. All the graphs in this section give the 25 percentile, median and 75 percentile values.

Make-before-break handoffs

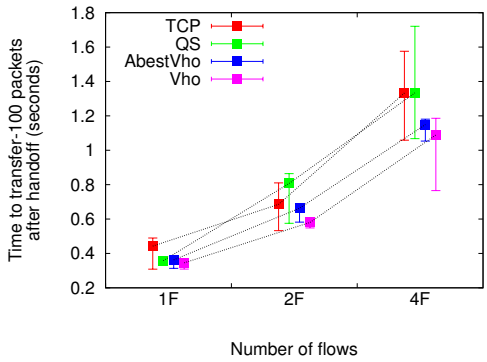

Figure 6: Time to transfer 100 packets after a make-before-break handoff from a 54000 Kbps/4 ms link to a 6000 Kbps50 ms link

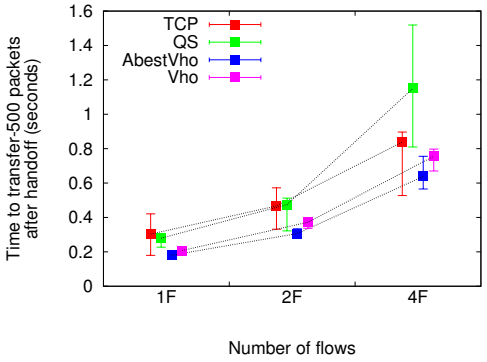

Figure 7: Time to transfer 500 packets after a make-before-break handoff from a 6000 Kbps/50 ms link to a 54000 Kbps/4ms link

Figures 6 to 9 show the transfer time for n packets after different make-before-break handoff scenarios. TCP and QS TCP are affected by the problems of vertical handoffs. Vho TCP mitigates the problems of TCP due to a vertical handoff and also sets the *ssthresh* to the BDP of the new access link after a vertical handoff. AbestVho incorporates the algorithms and estimates the available bandwidth of the end-to-end path for setting the *ssthresh*. All the above scenarios show the

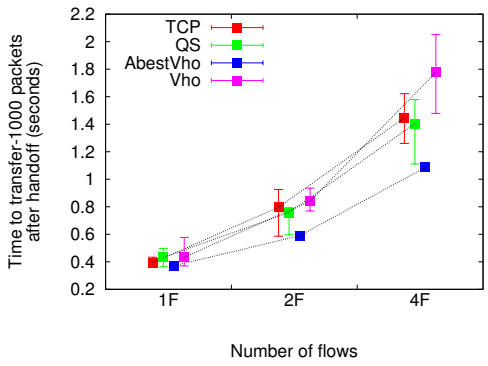

Figure 8: Time to transfer 1000 packets after a make-before-break handoff from a 54000 Kbps/4 ms link to a 50000 Kbps/15 ms link

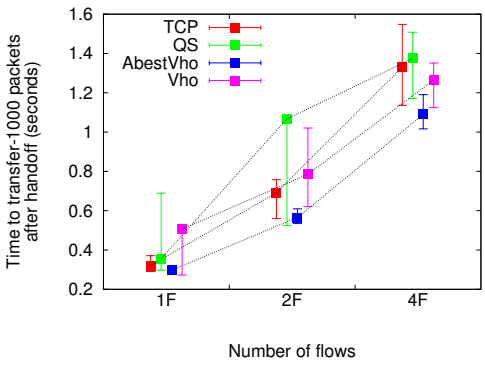

Figure 9: Time to transfer 1000 packets after a make-before-break handoff from a 50000 Kbps/15 ms link to a 54000 Kbps/4ms link

reduction in transfer time achieved by the Vho algorithms. In the multiple flow cases, Quick-Start has higher transfer time compared even to TCP due to the inaccurate setting of the initial rate in the case of multiple TCP connections.

Figures 6 and 7 show the make-before-break handoff between WLAN (54 Mbps/4 ms) and HSDPA (6 Mbps/ 50ms) where the BDP of the access links are 38 and 50 packets respectively. We can see from the above figures that when the handoff is to a slower link, 6 Mbps/50 ms link, Vho TCP and AbestVho performs in a similar way.

Figures 8 and 9 show the make-before-break handoffs between WLAN (54 Mbps/4 ms) and LTE (50 Mbps/ 15ms) where the BDP of the access links are 38 and 125 packets respectively. In both the handoff scenarios, AbestVho has at least 20 % reduction in transfer time compared to other TCP versions. The above figures clearly show the effectiveness of the bandwidth estimation algorithm together with the vho algorithms in the case of handoff to a high-bandwidth/high-BDP link.

Break-before-make handoffs

The make-before-break vertical handoff is yet to be a reality today. The usual practice in the cellular world is still a break-before-make handoff. So we had a study of break-before-make handoffs from EGPRS to LTE and HSDPA to LTE.

Figures 10 and 11 show two break-before-make handoff scenarios. The graphs show the reduction in transfer time achieved by the Vho algorithms. In addition to mitigating the problems of TCP due to a vertical handoff Vho TCP sets the *ssthresh* to the BDP of the new access link after a vertical handoff. AbestVho algorithms incorporating both Vho algorithms and bandwidth estimation methods achieves the minimum transfer time compared to other TCP versions.

Figure 10 shows the transfer time for a break-before-make handoff from EGPRS (200 Kbps/300 ms) to LTE (50 Mbps/15 ms), there is a significant increase in BDP from 10 packets to 125 packets. In the break period of one second in the above break-before-make handoff, TCP timer expires more than once and the unused connection time and the *ssthresh* reduction increases the transfer time for

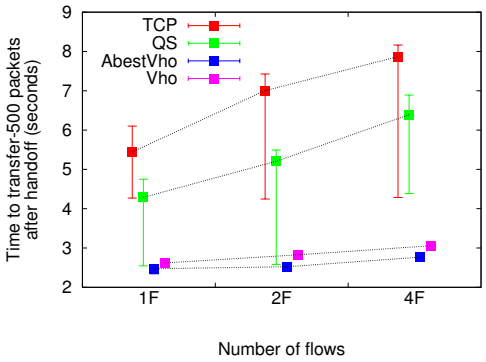

Figure 10: Time to transfer 500 packets after a break-before-make handoff from a 200 Kbps/ 300 ms link to a 50000 Kbps/15 ms link, disconnection period of 2 s

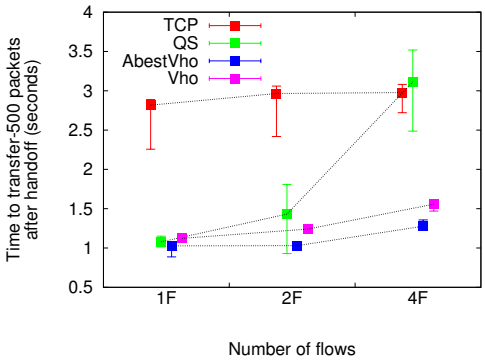

Figure 11: Time to transfer 500 packets after break-before-make handoff from a 2000 Kbps/50 ms link to a 50000 Kbps/15 ms link, disconnection period of 500 ms

TCP. QS TCP also suffers from the unused connection time, but the Quick-Start algorithm helps to reduce the transfer time. Here, Vho TCP and AbestVho reduces the transfer time by more than 50 %. AbestVho achieves a slightly lower transfer time than Vho TCP.

Figure 10 shows the transfer time for a break-before-make handoff from HSDPA (2000 Kbps/50 ms) to LTE (50 Mbps/15 ms). AbestVho achieves the smallest transfer time with an improvement more than a factor of 2.

4.3 After a disconnection

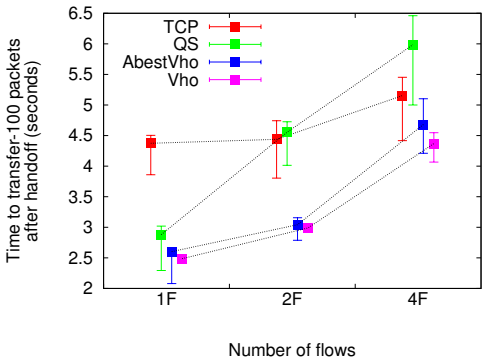

Figure 12: Time to transfer 100 packets after after a disconnection of 1 s in a a 2000 Kbps/50ms link

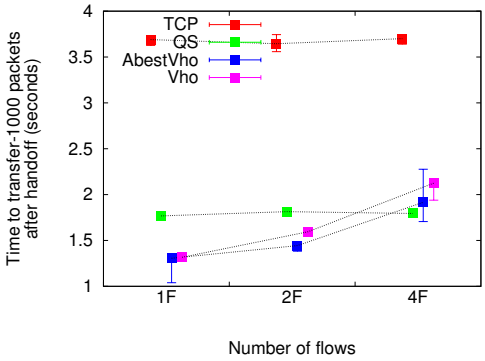

Figure 13: Time to transfer 1000 packets after a disconnection of 500 ms in a 50000 Kbps/50 ms link

In this set of experiments we evaluate the performance of TCP versions after a disconnection or link outage. The disconnection period is set as one second for HSDPA link and 500 ms for LTE link.

Figure 12 shows the transfer time when a disconnection of one second occurs in an HSDPA (2000 Kbps/50 ms) link. During the disconnection, TCP retransmission timer expires at least two times.

TCP and QS TCP suffer from unused connection time and whereas TCP repeatedly reduces the *ssthresh*. Vho TCP and AbestVho have the minimum transfer time compared to other TCP versions.

Figure 13 shows the transfer time for TCP versions when the connectivity breaks for 500 ms in an LTE link. AbestVho has the minimum transfer time compared to other TCP versions. It is interesting to note the behaviour of QS TCP in this scenario. The RTO value here is slightly higher than 500 ms and the retransmission timer expires immediately after the connection comes up. So QS TCP with the Quick-Start algorithm is able to set the *cwnd* and *ssthresh* values immediately after the disconnection period thereby having a similar performance as AbestVho. In break-before-make handoffs and disconnection scenarios Vho TCP and AbestVho immediately retransmit the first unacknowledged packet if the retransmission timer has expired during the disconnection.

The above experimental study points out the effectiveness of AbestVho in the start-up behaviour of TCP, in handoffs and in disconnection. AbestVho improves the TCP performance especially in high-bandwidth/high-BDP paths.

5 Conclusions and Future Work

In this paper we extend the vertical handoff (vho) algorithms we have proposed earlier with bandwidth estimation methods. Our simulation study in current wireless networks such as LTE, WLAN, HSDPA and EGPRS show that vho algorithms with bandwidth estimation are effective in setting the initial *ssthresh* of a TCP connection, in vertical handoff and in intermittent connectivity arising in wireless networks. In this paper we have compared the proposed algorithm with different TCP versions such TCP SACK, Quick-Start enabled TCP, TCP with initial window of 10 packets. It would be interesting to compare the proposed algorithm with RAPID and NF-TCP as both of these schemes are using available bandwidth estimation techniques to achieve congestion control. It is also worth to study how well RAPID and NF-TCP react to vertical handoffs. As a future study we would also like to evaluate our algorithms in real wireless networks.

Acknowledgements

This work was supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

References

[1] 3GPP. High Speed Downlink Packet Access (HSDPA). Technical Report 3GPP TS 25.950, July 2005.

[2] 3GPP. 3rd Generation Partnership Project; Long Term Evolution. Technical Report 3G TS 36 version, December 2008.

[3] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. Internet RFCs, ISSN 2070-1721, RFC 3042, January 2001.

[4] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. Internet RFCs, ISSN 2070-1721, RFC 3390, October 2002.

[5] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. Internet RFCs, ISSN 2070-1721, RFC 5681, September 2009.

[6] M. Arumaithurai, X. Fu, and K. K. Ramakrishnan. NF-TCP: A Network Friendly TCP Variant for Background Delay-Insensitive Applications. In *Proc. 10th IFIP International Conference on Networking (Networking 2011)*, May 2011.

[7] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgement (SACK)-based Loss Recovery Algorithm for TCP. Internet RFCs, ISSN 2070-1721, RFC 3517, April 2003.

[8] R. Braden. Requirements for Internet Hosts – Communication Layers. Internet RFCs, ISSN 2070-1721, RFC 1122, October 1989.

[9] Caida. Internet Traffic Classification. Available at: http://www.caida.org/research/traffic-analysis/classification-overview, 2009.

[10] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's Initial Window. Internet-Draft "draft-ietf-tcpm-initcwnd-03.txt", February 2012. Work in progress.

[11] L. Daniel and M. Kojo. Employing Cross-layer Assisted TCP Algorithms to Improve TCP Performance with Vertical Handoffs. *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, pages 433–465, November 2008.

[12] L. Daniel and M. Kojo. The Performance of Multiple TCP Flows with Vertical Handoff. In *Proceedings of the 7th ACM International Symposium on Mobility Management and Wireless Access (MobiWac'09)*, pages 17–25, October 2009.

[13] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-Start for TCP and IP. Internet RFCs, ISSN 2070-1721, RFC 4782, January 2007.

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

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

[16] J. Hoe. Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes. Master's thesis, Massachusetts Institute of Technology, June 1995.

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

[18] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988.

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

[20] S-E. Kim and J. A. Copeland. TCP for Seamless Vertical Handoff in Hybrid Mobile Data Networks. In *Proceedings of IEEE Globecom 2003*, December 2003.

[21] V. Konda and J. Kaur. RAPID: Shrinking the Congestion Control Timescale. In *Proceedings of IEEE INFOCOM 09*, 2009.

[22] D. Koutsonikolas and Y. C. Hu. On the feasibility of bandwidth estimation in wireless access networks. *Wireless Networks*, 17(6), August 2011.

[23] K. Lakshminarayanan, V.N. Padmanabhan, and J. Padhye. Proceedings of the 4th ACM SIGCOMM conference on Internet measurement. In *IMC 04*, 2004.

[24] M. Li, M. Claypool, and R. Kinicki. WBest: a Bandwidth Estimation Tool for IEEE 802.11 Wireless Networks. In *Proceedings of 33rd IEEE Conference on Local Computer Networks (LCN)*, 2008.

[25] J. Manner and M. Kojo (Eds.). Mobility Related Terminology. Internet RFCs, ISSN 2070-1721, RFC 3753, June 2004.

[26] Network Simulator ns-2. http://www.isi.edu/nsnam/ns.

[27] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. Internet RFCs, ISSN 2070-1721, RFC 2988, November 2000.

[28] J. Postel. Transmission Control Protocol. Internet RFCs, ISSN 2070-1721, RFC 793, September 1981.

[29] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. Internet RFCs, ISSN 2070-1721, RFC 3168, September 2001.

[30] V.J. Ribeiro, R.H. Riedi, R.G. Baraniuk, J. Navratil, and L. Cottrell. PathChirp: Efficient Available Bandwidth Estimation for Network Paths. Passive and Active Measurement Workshop, PAM2003, April 2003.

[31] P. Ruuska, J. Mäkelä, M. Jurvansuu, J. Huusko, and P. Mannersalo. ROADMAP for Communication Technologies, Services and Business Models 2010, 2015 and Beyond. http://www.tekes.fi/u/GIGA-Roadmap2010.pdf, August 2010.

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

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

[34] E. Seurre, P. Savelli, and P.-J. Pietri. *EDGE for Mobile Internet*. Artech House, 2003.

[35] A. Shriram and J. Kaur. Empirical evaluations of techniques for measuring available bandwidth. In *Proceedings of IEEE INFOCOM 07*, 2007.

[36] K. Sripanidkulchai, B. Maggs, and H. Zhang. An Analysis of Live Streaming Workloads on the Internet. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC'04)*, pages 41–54, 2004.

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