Performance Evaluation of TCP Enhancements over Slow Wireless Links

Pasi Sarolahti

Department of Computer Science, University of Helsinki

P.O. Box 26 (Teollisuuskatu 23),
FIN-00014 UNIVERSITY OF HELSINKI, Finland

E-mail: `sarolaht@cs.helsinki.fi`

Abstract

Increasing number of the hosts connected to the Internet are behind a slow wireless link. Most of the networking applications in the Internet use TCP protocol, which is known to have problems when used over wireless link. Recently, a number of enhancements have been suggested for TCP over wireless links. We will study the performance effects of using the SACK, limited transmit and the NewReno TCP with increased initial window. Additionally, we experiment with NewReno TCP with a limited, shared advertised window, which we expect to reduce the number of congestion losses on the bottleneck wireless link. We present the results of the performance tests made with the above-mentioned TCP enhancements when the link is free from other traffic, and under a heavy congestion on the wireless link.

1 Introduction

The number of wireless Internet hosts is likely to increase considerably in the near future. In addition to the hosts with a wireless LAN connection, Internet protocols will be used in wireless terminals operating in the *Wireless Wide Area Networks (W-WANs)*, such as GSM, GPRS and the third generation technologies. Using networking applications over wireless links is under a large amount of research, because the wireless links are prone to transmission errors and they usually have significantly lower bandwidth than the traditional fixed networks.

The majority of the Internet applications use *Transmission Control Protocol (TCP)* [16] as their transport layer protocol. TCP provides flow control and reliable delivery of the data segments between the communicating endpoints while adjusting the sender rate to avoid congesting the network. TCP assumes that if a packet is lost in the network, it is because a congested router has dropped it. Therefore TCP sender slows down its transmission rate when it observes a packet loss.

On a wireless link a packet can be lost because of transmission error caused by the natural environment. After such loss slowing down the transmission rate is a wrong action to take. A significant amount of TCP research is targeted on improving TCP performance when communicating over wireless link (see e.g. [14]).

The wireless link layer protocol can offer reliable service to the upper layer protocols by retransmitting the missing link-layer frames. If a reliable link layer protocol is used, TCP protocol sees the transmission errors as delays of variable length instead of packet losses. Using reliable link layer protocol leads generally to better performance than doing end-to-end retransmissions by the TCP protocol [5]. Throughout this paper we assume a reliable link layer protocol.

There are two main approaches that can be applied at the TCP layer to improve the performance when communicating over wireless links. Many solutions use *proxies* located between the wireless link and the fixed Internet [6, 11, **?**]. By using proxies it is easy to differentiate the packet losses that occur on the wireless link from the packet losses that occur in the fixed network, making it possible to improve the performance of wireless TCP communication. However, use of proxies can violate the end-to-end semantics of the Internet [14] used as a main design principle of the Internet protocols. This can be harmful for the functionality and interoperation of some Internet protocols, such as *IPsec* [9].

Because proxies have their disadvantages, it is desireable to aim for performance improvements by modifying the behaviour of TCP endpoints without having to affect the behaviour of the network hosts in the middle of the connection path. Therefore, we study different modifications of the

TCP implementation at the connection endpoints. Some TCP enhancements may require modification at the both ends of the TCP connection, whereas others alter the behaviour of only one endpoint while preserving the compatibility with the legacy TCP implementations at the other end of the connection.

The rest of this paper is organized as follows: In Section 2 we describe the network architecture typically used in W-WAN networks. Section 3 illustrates certain specific problems affecting the TCP performance. The TCP enhancements we are evaluating in this study to overcome the described problems are introduced in Section 4. We describe the test arrangements of the performance tests in Section 5 and Section 6 gives the results of the performance tests made. Finally, we conclude our work in Section 7.

2 Wireless Network Architecture

We assume an architecture illustrated in Figure 1 for the wireless networking environment under study. The figure shows the mobile host that communicates with the base station over the wireless link. A last-hop router forwards the packets between the base station and the fixed Internet. This model is similar to the architecture used in most W-WAN systems.

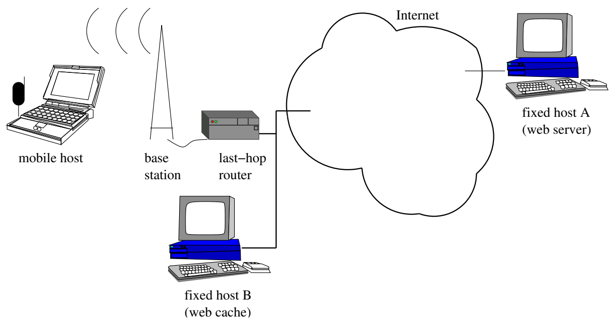

Figure 1: W-WAN network architecture.

The mobile host communicates with the base station over a wireless link. The available bandwidth on W-WAN wireless links is considerably slower than the bandwidth usually available in the fixed network. For example, the GSM data service [15] provides bandwidth of 9600 bps. The latencies on W-WAN wireless links vary between 100–300 ms. Therefore the resulting bandwidth-delay product on the link is not necessarily very high[1].

W-WAN systems often use a reliable link layer protocol, which retransmits the missing frame until it is succesfully delivered. Because packet reordering is usually harmful for transport layer protocols, especially TCP, the link layer receiver stores the incoming frames in its buffers until the packets can be delivered to the upper protocol layers in right order. This behaviour causes variable delays in packet transmissions.

The packets usually arrive at the last-hop router from the fixed network at a faster rate than it can forward them to the wireless link. Therefore the input queue length at the router is about to grow very easily. Because queueing increases the packet delays, the last-hop router queue length is usually limited to 3–4 packets[2].

Usually most of the traffic is transmitted downlink, that is, from the server located in the fixed network towards the mobile host. This is the case e.g. with web transfers, where usually only small requests are transmitted in uplink direction. The Internet Servive Providers (ISPs) often use a web cache in their local network to reduce the amount of packets needed to transfer to the public network. If the requested information is available in the local network, the transmission path for the packets tends to be very simple, consisting only one or few hops in addition to the wireless link.

3 TCP Problems over Slow Wireless Links

Earlier studies have reported different problems in TCP performance when it is used over slow or unreliable links. In this section we briefly introduce

[1] Bandwidth-delay product indicates the maximum amount of data that can be outstanding on the link at once.

[2] We assume that each mobile user has dedicated channel on the link and a dedicated space in the last-hop router. This is how e.g. GSM allocates the link and the last-hop buffers.

the main problems we expect to face in our performance tests.

We use *NewReno TCP* [8] as the baseline TCP implementation in our study. In addition to the traditional *slow start* and *congestion avoidance* algorithms, NewReno TCP sender uses *fast recovery* algorithm for recovering packet losses. During fast recovery the TCP sender can transmit new data while waiting for acknowledgements of retransmissions. When the TCP sender receives three duplicate acknowledgements (ACKs), it does a *fast retransmit* for the first unacknowledged segment. Fast retransmit was introduced to avoid *retransmission timeout(RTO)*, which usually causes idle periods in transfer and is therefore harmful for the performance. After the fast retransmit the sender is in fast recovery until it has succesfully received acknowledgements for all segments that were lost in the last window of transmissions.

Slow start overshooting Initially, the TCP sender starts in slow start state, increasing the *congestion window (cwnd)* by one for each incoming acknowledgement. The sender exits the slow start to the congestion avoidance state when the congestion window size reaches the *slow start threshold (ssthresh)*. Because `ssthresh` is initialized to an infinite value, the initial slow start is exited after a packet is lost for the first time. As the link layer is reliable, the packet loss occurred because the last-hop router queue was filled up and it had to drop the incoming packet. However, the TCP sender observes the packet loss from the duplicate ACKs generated by the receiver only roughly after one round-trip time from the first packet loss has passed. During this time the sender has transmitted one window of data still using the slow start algorithm to a full last-hop router queue. As a result, there usually are several packets lost during the time from the first packet loss to the arrival of the duplicate ACKs at the sender. This behaviour is called *slow start overshooting* [12], and it is illustrated in the time-sequence diagram in Figure 2.

Spurious timeouts The delay for transferring packets over the wireless link is likely to vary substantially (from 200 ms to several seconds) for various reasons, such as link layer retransmissions or hand-offs. In some cases a suddenly increased packet delay is high enough to trigger the TCP RTO at the sending end. As a result the TCP sender reduces its *congestion window* to one segment and retransmits the whole window of data from

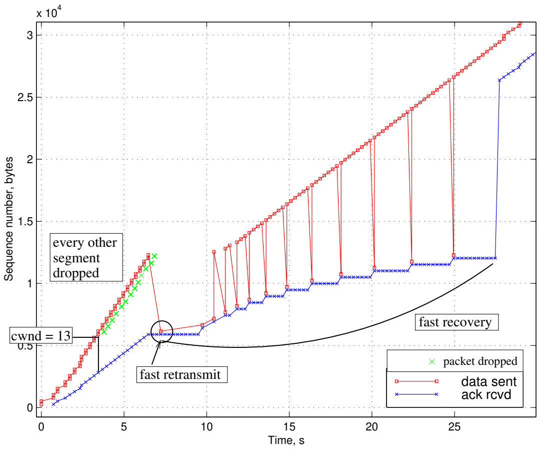

Figure 2: Slow start overshooting.

the first unacknowledged packet. This is called go-back-N behaviour. As the RTO was caused by excessive delay, most of the retransmissions are unnecessarily made, hence the scarce bandwidth is unnecessary wasted. Additionally, the unnecessary retransmitted segments appear to the receiver as out-of-order segments which then generates duplicate ACKs. These, in turn, unnecessarily trigger the fast retransmission and cause the sender to enter the fast recovery algorithm. Figure 3 illustrates a spurious retransmission timeout with its consequences.

4 Suggested TCP Enhancements

We now briefly introduce the selected TCP enhancements that have been suggested to improve the TCP performance over links with high latency or a high packet loss rate. We evaluate the performance implications of using the TCP enhancements presented below over an emulated slow wireless link.

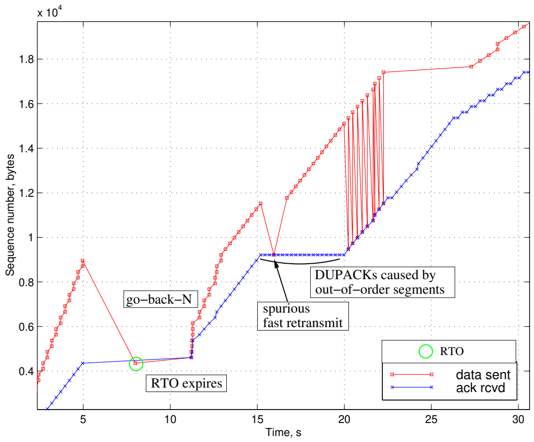

Figure 3: Spurious retransmission timeouts.

Selective Acknowledgements (SACK) [13] Because the basic TCP uses cumulative acknowledgements, a TCP receiver can only acknowledge the highest segment received in a continuous stream of data. If a segment is missing, the segments following it cannot be acknowledged before the missing segment is successfully retransmitted. As a result, the TCP sender can only do one retransmission in a round-trip time. SACK was introduced to overcome this problem by using TCP option headers to acknowledge up to four blocks of data in addition to the cumulative acknowledgement. This makes it possible for the sender to retransmit more than one missing segment during round-trip time, thus improving TCP performance after slow start overshooting, for example. The SACK option handling needs to be implemented at the both ends of the TCP connection.

Limited transmit [2] If the packet loss rate is very high, the TCP congestion window tends to be small. Because TCP sender requires three successive duplicate ACKs to trigger fast retransmit, congestion window smaller than four segments makes the fast retransmit and fast recovery algorithms unusable. By following the limited transmit algorithm the TCP sender is allowed to transmit a new segment for each arriving duplicate ACK before the fast retransmit takes place. Because the receiver generates a new duplicate ACK for each incoming out-of-order segment, the sender will eventually get three duplicate ACKs to trigger the fast retransmit. We will also evaluate the effect of combining SACK and limited transmit, because the SACK retransmissions also depend on the incoming duplicate ACKs.

Increased initial window [3] Traditionally a TCP sender starts with initial congestion window size set to one or two segments, and then follows the slow start algorithm to increase the window size. If the path between the communicating TCP end hosts has a large bandwidth-delay product, it takes several round-trip times to increase the congestion window to a size that corresponds to the network capacity. Therefore, using larger initial congestion window (3–4 segments) has been suggested to speed up the startup of the connection. Obviously, such a modification can increase the level of congestion in the network, and cause more packets to be lost. We will evaluate the performance effects of using a larger initial window when used over slow wireless link.

Limited advertised window Avoiding the slow start overshooting and the related packet losses is an attractive goal, as it obviously would improve the TCP performance. However, the TCP sender is often a legacy server in the Internet that is unable to know the path characteristics between the client and the server, and thus unable the guess a proper value for the congestion window size in the beginning of the connection. However, we assume that the mobile user usually knows the approximate characteristics of the last-hop link he is using. Because the TCP receiver is usually the mobile host, limiting the sending rate at the receiver is reasonable. This is possible by setting an upper bound for the TCP advertised window size that controls the maximum number of packets outstanding in the network. We introduce a modification of TCP that uses a defined upper limit for the advertised window size. The advertised window limit is shared among all TCP connections using the wireless link and scaled according to the number of connections to keep the number of outstanding packets below the estimated buffering capacity of the last-hop router and the wireless link.

5 Test Arrangements

We evaluate the performance of the selected TCP enhancements in an environment that consists of a fixed end server attached to a LAN, a last-hop router in the same LAN, and a 9600 bps wireless link between the last-hop router and the mobile host. Upper part of Figure 4 illustrates the environment we are modelling in our study. We believe the illustrated arrangement corresponds to a setup often used with the corporate ISPs, when a web proxy in the local network is used.

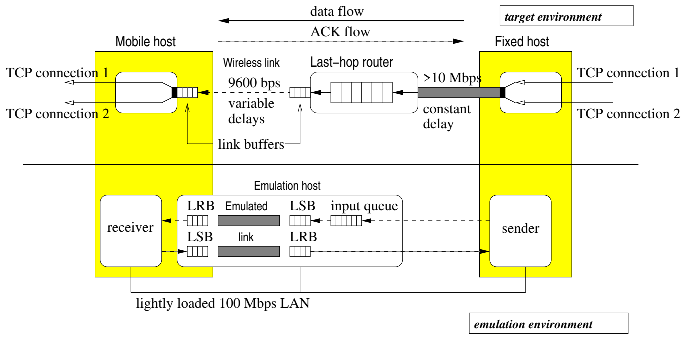

Figure 4: The test environment.

We use a real-time network emulator called *Seawind* [10] to simulate the wireless link and the last-hop router. Seawind captures the packets from the network interfaces of the end hosts transparently to the IP implementation and the protocols above it, making it possible to evaluate the protocol implementations of real operating systems. The slow, high-latency link is emulated by delaying the network packets appropriately. Link-layer retransmissions are emulated by holding the affected packet a specified amount of time before delivering it onward. Although this is a simple way of emulating link layer retransmissions, we believe it is sufficient in order to study the effects of excessive link layer delays on the

TCP protocol. Because extensive link-layer delays can occur for many reasons, we don't want to restrict the source of the delays too much.

We use Linux kernel version 2.4 at the end hosts in our performance tests. Linux version 2.4 already has SACK[3] and limited transmit algorithm implemented. We made some additional modifications in the Linux kernel to increase the initial window size and to limit the advertised window size. The Seawind simulation process was located in a third Linux host that was free from other use to ensure accurate response times in emulation. All three hosts were located on a isolated 100 Mbps LAN that was free from network traffic not related to the emulation. The bottom part of Figure 4 illustrates the test setup.

We evaluate the TCP enhancements by measuring the connection times and studying the detailed behaviour of the TCP variants in a few test scenarios described below. We use unidirectional bulk transfers originating from the fixed end host as the workload. We have two workload models: 100 KB transferred in a single TCP connection, and four parallel TCP connections each transferring 50 KB of data. We measure the time elapsed from sending the first TCP SYN segment to receiving the last TCP FIN acknowledgement at the TCP sender[4]. In addition, we measure the time used in transmitting the first 10 KB of the data in each TCP connection. This metric emphasizes the effect of the initial TCP slow start. Furthermore, most of today's TCP connections transfer 10 KB or less [1].

We model a last-hop router queue of 3–7 packets with drop-tail policy. In addition to the router's input queue we model link buffers of 1220 bytes at the both ends of the wireless link. The link buffer is needed at the sending end to store link layer frames that are not yet acknowledged in order to retransmit lost frames. At the receiving end the link buffer stores out-of-order frames to be able to forward the packets in right order to the upper protocol layers. The link buffers can hold up to 4 packets worth of data in addition to the data in the router queue. The link bandwidth is 9600 bps and the propagation delay over the link is 200 ms. The TCP *Maximum Segment Size* is 256 bytes.

We use two link models in our tests. In the first test case we don't

[3]Linux SACK follows the FACK retransmission algorithm [12].

[4]SYN segment is used for the three-way handshake in the beginning of a TCP connection. FIN segment is used to finish the TCP connection.

have any additional delays on the link, except the packet transmission delay and the propagation delay. In the second scenario a 6-second additional delay can occur on a packet with the probability of 0.01. The delay length of 6 seconds was fixed, because it is usually long enough to trigger a TCP RTO, and we wanted to study the effects of having RTOs at the different stages of the TCP connections. Moreover, our preliminary tests show that short delays not causing the retransmission timer to expire do not affect significantly on TCP performance. Each test case is repeated 20 times with different random values. However, the same random distribution is used when testing the different TCP enhancements to improve the comparability of the TCP variants.

6 Results of the Performance Tests

This section presents the results of the performance tests with the different TCP enhancements. Table 1 shows the results of tests made a single TCP connection using a last-hop router buffer size of 3 packets. The table shows median times of 20 test replications made with and without the excessive 6-second random delays. In addition to the time used for the whole 100 KB connection, the table shows also the time used for transmitting and getting acknowledgements for the first 10 KB of the workload data.

Table 1: Elapsed times for tests with a single TCP connection.

TCP variant	**No excessive delay**		**6 s. delay, pr. 0.01**	
	Time (s) / 10 KB	**Time (s) / 100 KB**	**Time (s) / 10 KB**	**Time (s) / 100 KB**
NewReno	17.30	108.24	17.29	137.88
SACK	11.60	101.94	11.31	136.10
Limited xmit	17.91	107.43	14.80	131.65
SACK + Lim. xmit	11.59	100.42	11.31	129.00
IW = 4	17.45	107.83	17.09	138.12
Adv. wnd = 2 KB	11.29	100.42	11.31	132.91

Table 1 confirms some of the expectations made earlier. SACK TCP performs clearly better than the NewReno TCP in the beginning of the connection, when the slow start overshooting occurs. Because by limiting the receiver's advertised window the packet losses due to slow start overshooting can be avoided, the performance is ideal with the limited advertised window ($Adv.\ wnd$ = $2KB$). Limited transmit and increased initial window (IW = 4)do not have significant impact on TCP performance when there are no competing traffic on the wireless link. When excessive delays are present, combination of SACK and limited transmit give the best performance.

Table 2 shows the results of tests made with four parallel bulk TCP connections. There were no excessive delays on the results shown in addition to the transmission and propagation delays. The table shows the time used to transmit and get acknowledgements for the first 10 KB of the workload data when using a last-hop router buffer size of 3 and 7 packets. The medians of 20 measurements for the fastest and the slowest connection are shown. From the performance difference of the fastest and the slowest TCP connection we can get a primitive measure of fairness between the connections. We are primarily interested in the time of the slowest connection, because that indicates when all transfers are finished.

Table 2: Elapsed times for tests with four parallel TCP connections. First 10 KB of the TCP connections shown.

TCP variant	**3-pkt buffer**		**7-pkt buffer**	
	Time (s) / fastest	**Time (s) / slowest**	**Time (s) / fastest**	**Time (s) / slowest**
NewReno	30.74	102.46	26.89	150.51
SACK	20.93	65.29	27.37	108.76
Limited xmit	23.67	72.65	33.26	55.72
SACK + Lim. xmit	30.26	131.29	29.28	81.66
IW = 4	21.39	138.58	24.26	192.99
Adv. wnd = 3 KB	24.47	161.72	29.25	48.94

Again, the table confirms the expected result of SACK TCP performing better than the NewReno TCP. With four parallel TCP connections the packet losses due to congestion occur frequently, which obviously promotes the use of SACK TCP being able to do more than one retransmission in a round-trip time. Use of limited transmit improves the performance, because the high packet loss rate keeps the congestion window of the TCP connections small. Without limited transmit there are a large number of TCP RTOs, whereas use of limited transmit decreases the number of RTOs, which improves the TCP throughput considerably.

Combining SACK and limited transmit gives worse performance with four parallel connections than using limited transmit or SACK alone, especially with 3-packet last-hop router buffers. A more detailed look to the test results reveals that the combination of SACK and limited transmit increases the number of congestion losses. This shows the sensitivity of TCP on heavy congestion situations. Because the FACK algorithm assumes that all segments between the highest received SACK acknowledgement and the cumulative acknowledgement have left the network, it starts retransmitting slightly earlier than the traditional congestion control methods. The four parallel TCP connections can tolerate the increased aggressiviness of the FACK algorithm or the limited transmit separately, but when combined, the slightly higher transmission rate in the beginning of the fast recovery causes the router queue to be occupied constantly. This causes some of the connections to suffer from increased number of RTOs due to losses of consequtive segments.

When increasing the initial congestion window to four segments, some of the TCP connections are unable to succesfully transmit any segments to the receiver in the beginning of the connection. The other TCP connections use the available capacity on the link and in the last-hop router buffer, taking the unfortunate connections to several consequtive backed-off RTOs. This problem is also identified in related studies and it is commonly called the *lock-out problem* [7]. Because of the lock-out behaviour, the last TCP connection takes considerably more time to finish than with the initial window of two segments, when the lock-out problem is not as severe.

The results of the shared advertised window tests with window of 12 segments show that when the advertised window size is appropriately set, the fairness is significantly improved, which results in clearly better throughput of the slowest connection. By using the small advertised window most of the congestion losses are avoided. The results also show that if the advertised window size is set incorrectly (i.e. 12 segments for 3-packet buffers), the performance can actually be worse than without the advertised window limitation. In the 3-buffer case the advertised window of 12 segments is not small enough to avoid the packet losses, but the smaller window restricts the proper use of the fast recovery algorithm to recover from the packet losses.

7 Conclusion

We presented the results of the performance tests made using different TCP variants over emulated slow wireless link. As expected, use of SACK improved the TCP performance. We also presented the benefits of limiting the receiver's advertised window and sharing it among the TCP connections used over the wireless link. When four parallel TCP connections were used, increasing the initial congestion window increased the level of congestion on the wireless link which caused degradation in the performance due to larger number of packet losses. Similarily, combining SACK and limited transmit kept the last-hop router buffer full with four TCP connections, which resulted in poor throughput with the slowest TCP connection. Although heavy congestion was present with these TCP variants, later tests show that on a higher bandwidth of 28800 bps the congestion does not cause as much problems, as it did on the 9600 bps link used in this study.

In future we will combine the useful TCP enhancements and try to tune them further to find out the optimal settings. Inspecting the methods for evaluating the link bandwidth-delay product at the receiving end and then dynamically adjust the shared advertised window size is worth of further efforts. We will also study the performance of different workload models in addition to the basic models introduced here, such as HTTP transfers and real-time streaming data.

References

[1] M. Allman. A Web Server's View of the Transport Layer. *ACM Computer Communication Review*, 30(5), October 2000.

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

[3] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. Internet draft "draft-ietf-tsvwg-initwin-00.txt", May 2001. Work in progress.

[4] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 136–143. IEEE, May 1995.

[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *Proceedings of ACM SIGCOMM '96*, Stanford, CA, August 1996.

[6] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. IETF RFC 3135, June 2001.

[7] Braden, et. al. Recommendations on Queue Management and Congestion Avoidance in the Internet. IETF RFC 2309, April 1998.

[8] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 2582, April 1999.

[9] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. IETF RFC 2401, November 1998.

[10] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen. Seawind: a Wireless Network Emulator. In *Proceedings of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, RWTH Aachen, Germany, September 2001. To appear.

[11] M. Kojo, K. Raatikainen, M. Liljeberg, J. Kiiskinen, and T. Alanko. An Efficient Transport Service for Slow Wireless Links. *IEEE Journal on Selected Areas In Communications*, 15(7):1337–1348, September 1997.

[12] M. Mathis and J. Mahdavi. Forward acknowledgement: Refining TCP Congestion Control. In *Proceedings of ACM SIGCOMM '96*, October 1996.

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

[14] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. Long Thin Networks. IETF RFC 2757, January 2000.

[15] M. Mouly and M. Pautet. *The GSM System for Mobile Communications*. Europe Media Duplication S.A., 1992.

[16] J. Postel. Transmission Control Protocol. IETF RFC 793, September 1981.